8-1995

# Addition of Automatic Dimensioning Feature to D-M-E’s Ultimate Mold Base Design Software

Cori L. Brown

## Recommended Citation

Brown, Cori L., "Addition of Automatic Dimensioning Feature to D-M-E’s Ultimate Mold Base Design Software" (1995). *Master's Theses*. 4901.
https://scholarworks.wmich.edu/masters_theses/4901

# ADDITION OF AUTOMATIC DIMENSIONING FEATURE TO D-M-E'S ULTIMATE MOLD BASE DESIGN SOFTWARE

by

Cori L. Brown

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Industrial and
Manufacturing Engineering

Western Michigan University
Kalamazoo, Michigan
August 1995

# ACKNOWLEDGMENTS

I find it a challenge to find the words to express my gratitude and appreciation to my mentor, Dr. Michael Atkins. He has been both an inspiration and a great support to me throughout my entire thesis project, as well as, all my years at Western Michigan University. I would also like to thank Dr. Paul Engelmann for having confidence in my abilities every step of the way. He always knew when I needed a boost . I also want to thank Dr. Ralph Tanner for his continued assistance and knowledge throughout my project.

Another person that I want to thank is my friend Jim Ignatovich. He has made it possible for me to carry on my educational career no matter what the obstacle. And last but not least, I would like to thank my family. They helped keep me sane and were always there when I needed them, even though they never truly understood what I was doing.

Thanks to all of you.

Cori L. Brown

# ADDITION OF AUTOMATIC DIMENSIONING FEATURE TO D-M-E'S ULTIMATE MOLD BASE DESIGN SOFTWARE

Cori L. Brown, M.S.

Western Michigan University, 1995

There have been many improvements in the plastics industry over the past few years. One of these areas is in the design of injection molds. This area has been enhanced by the use of Computer-Aided Design systems and furthered by the creation of specialized mold design software. The Ultimate® design system from D-M-E has made a mold designer's job much easier by automating much of this process. This project extends the function of this software by adding an automatic dimensioning feature that will make the mold design process less tedious and less time consuming. This addition will also standardize mold drawing layouts created in this software and for industry as a whole.

# TABLE OF CONTENTS

ACKNOWLEDGMENTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ii

LIST OF FIGURES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . vi

CHAPTER

I. INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

- Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
- Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . 4
- Significance of the Problem . . . . . . . . . . . . . . . . . . . 5
- Expected Results . . . . . . . . . . . . . . . . . . . . . . . . . 6
- Assumptions . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
- Limitations . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

II. REVIEW OF LITERATURE . . . . . . . . . . . . . . . . . . . . . . . . 8

- Application Development . . . . . . . . . . . . . . . . . . . . . 8
- Programming Languages . . . . . . . . . . . . . . . . . . . . . . 12
- Mold Design . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
- Dimensioning Standards and Practices . . . . . . . . . . . . . . . 18

III. METHODOLOGY . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

- Preliminary Review . . . . . . . . . . . . . . . . . . . . . . . . 21
- Compilers . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

Table of Contents - Continued

CHAPTER

Development of Test Programs . . . . . . . . . . . . . . . . . . . . . . . . . 23
Background Information . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
Standard Dimensions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
Final Mold Base Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
IV. SOFTWARE DESIGN AND DESCRIPTION . . . . . . . . . . . . . . . . . . 31
Preliminary Review . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
Compilers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
Development of Test Programs . . . . . . . . . . . . . . . . . . . . . . . . . 32
Background Information . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
Standard Dimensions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
Final Mold Base Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
V. CONCLUSIONS AND RECOMMENDATIONS . . . . . . . . . . . . . . . . . 48
Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
Recommendations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49

APPENDICES

A. Ordinate Dimensioning Example . . . . . . . . . . . . . . . . . . . . . . . 51
B. Gant Chart . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53
C. Tabulated Information From Survey . . . . . . . . . . . . . . . . . . . . . 55

Table of Contents - Continued

APPENDICES

D. Initial Programs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

E. D-M-E Mold Layout . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81

F. Data File Format . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83

G. Ultimate® Layout (Four Views) . . . . . . . . . . . . . . . . . . . . . . 88

H. Survey Letter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90

I. Final Program . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93

J. Final Mold Base Layout . . . . . . . . . . . . . . . . . . . . . . . . 115

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117

## LIST OF FIGURES

1. Dimensioned Rectangle . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

2. Rectangle and Circle Graphics With Dimensions . . . . . . . . . . . . . . . . . . 36

3. Standard Dimensioning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

4. Ordinate Dimensioning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

5. Format of External Data File . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38

6. Schematic of Testing Procedures . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

# CHAPTER I

# INTRODUCTION

## Background

The area of computer graphics has exploded over the past decade and the applications are almost limitless. The creation of Computer-Aided Design (CAD) software has greatly enhanced the engineering and technical field today. There are several advantages in using CAD systems in design, these include: (a) easier creation and correction of working drawings, (b) easier visualization of drawings, (c) ease of reference for modification, (d) quick and convenient solution of computational design analysis problems, (e) simulation and testing of designs, and (f) increased accuracy (Earle, 1991). The majority of CAD systems are used only as electronic drawing boards and not as true design and modelling systems (Taylor, 1992). The advanced capabilities of CAD, as compared with some of the less advantageous capabilities available to the designer, such as electronic drawing boards, are the focus of this project since it is in this arena where the true advantages are recognized.

Optimization CAD tools are used for a variety of applications. For example, finite element analysis (FEA) packages provide some form of shape and structural optimization. This will include stress calculations when a force is applied to certain areas of the part and deflection calculations by evaluating the surface of that body. There are other analysis techniques, such as, the boundary element method (BEM) for analyzing a surface or boundary of a part. This type of analysis, BEM, usually requires less time for calculations than the FEA process (CADKEY, 1993).

Another example of an optimization CAD tool is that of mold design software for the injection molding industry. There are a several packages that are available for injection mold designers. One example is D-M-E Company's Ultimate® software. This package can be classified as an advanced CAD tool because it is an intelligent system. In this case, the intelligent system will check to ensure that an added component of the mold design is suitable and consistent with the mold designed up to this point. If there are any errors or conflicts with other components in the design, the software will immediately alert the user to enable them to make the necessary changes to the mold design (Tecnocad Ltd., 1992).

Until recently, mold designers were only able to use the CAD system for duplication of mold bases and components from supplier catalogs, which was very time consuming. Tecnocad Ltd., located in Sligo, Ireland, recognized this problem and decided to do something about it. The designers and programmers at Tecnocad Ltd. composed a detailed outline of this new concept and the components that should be included in this new software package (Taylor, 1992).

Their crucial goals included that the software must be able to operate independently of the computer hardware platform and operating system and should also operate independently of the CAD system. In order to fulfill these two goals they decided to use the C general purpose programming language. Upon further inspection of the desired mold design system, it was established that there were other important aspects that needed to be met. These included:

1. Operation by the user should be easy and intuitive.
2. The modeling of the mold should be as complete as possible.
3. Many output forms should be available and automatic, these include, general assembly drawings, detail drawings, bill of materials, and output to a file for ease of ordering materials and cost estimation.
4. Components from different suppliers must be available and intermixing of all of them should be possible.

The software package that Tecnocad Ltd. created was called CAMold (The CAMold software is also marketed by the name Ultimate®). CAMold has the designer work in a CAD systems 2D environment, but the database that is stored is a full 3D, topological, feature-based model. This model is stored separately from the CAD system, but it has links to the CAD system so that designers are able to utilize the capabilities of the standard CAD system as well (Taylor, 1992).

The CAMold software is able to run on various CAD systems. One such system is CADKEY version 6.0. CAMold is able to automatically create the specific mold design chosen by the user in the CADKEY database and is also able to take advantage of the CADKEY user interface for communication to and from the user (Taylor, 1992).

CAMold also runs on a variety of computers. The computers in which CAMold can be run are a 386 or 486 Personal Computer (PC) with the DOS operating system, Sun Sparc stations, IBM RS6000, HP 9000 models 500,400, and 700, SGI Indigo and Iris, DECstation 5000, and Apollo Domain (Taylor, 1992).

The D-M-E Company has offered this software, CAMold, under the name Ultimate® for the U.S. market. The two packages are the virtually the same except that one is for the European market and one for the U.S. market. At this time, the Ultimate® mold design system is only available on the PC and with the CAD packages CADKEY and Autocad only.

The D-M-E Company plans on updating their mold design software to include a Windows application on the PC as well as a Windows package for the Unix operating system with CADKEY. They also want to make the output of the mold base a true 3D solid or wire frame model since it only outputs 2D layout drawings at this time. With the inclusion of these features, the Ultimate® software will become an invaluable tool for mold designers in the injection molding industry.

## Problem Statement

One of the problems in the injection molding industry today is that, when creating a mold base in a CAD environment, it is very time consuming to detail and dimension the entire mold after is has been designed. Other problems arise as well, such as, missing dimensions on the layout drawing and a lack of standardization of these detailed drawings when it comes to creating the final layout design.

## Significance of the Problem

Since the onset of CAD, designers of plastics parts have had increased productivity when creating models and detail drawings (Forbes, 1989). For the injection molding industry, mold and part analysis have become a true advantage to all molders. But up to this point there has been very few software packages developed specifically for injection mold base design. In the injection molding industry, there is a great demand for software that can automatically create a standard injection mold base (Mold Makers...,1993).

The D-M-E Company has created a software package that will automatically create a standard D-M-E mold base. Unfortunately, D-M-E's Ultimate® software package does not accommodate a major need of the mold designer. The most time consuming step of designing a mold is dimensioning the layout once the mold has been created on the computer. According to designers at Tandy Electronics Tool Engineering, "Dimensioning has proven to be another time-draining job, even when using some CAD packages" (Mold Makers..., 1993).

There are several other problem areas in the mold design industry that would benefit from an automatic dimensioning feature in this software. These include, standardization of mold drawings in industry and the possibility of missing dimensions would become obsolete.

## Expected Results

When the automatic dimensioning feature has been added to The D-M-E Company's Ultimate® mold base design software, the following goals will have been completed.

1. All dimensions will be determined per American National Standards Institute (ANSI) standards and by mold design practices in the injection molding industry.

2. Successful completion of compiled C program for the purpose of automatic dimensioning of the mold base created by the user within the Ultimate® software.

3. Successful completion of integrating the C compiled program into the original Ultimate® program to inspect the execution of the automatic dimensioning feature.

## Assumptions

This study will be developed under the following basic assumptions:

1. Tecnocad Ltd. will provide the necessary 'hooks' into the existing

software so that an executable program can be developed incorporating the added feature.

2. Tecnocad Ltd. will provide variable names or function names from the original program for positions on mold base for dimensioning purposes.

3. Tecnocad Ltd. will provide pieces of the original Ultimate® source code if necessary to complete this project.

## Limitations

The basic limiting factors pertaining to this study are as follows:

1. The Ultimate® software will create standard and customized D-M-E mold bases by given user dimensions.

2. This added feature for the Ultimate® software will be created only for CADKEY version 6.0.

3. The application software created will only run in CADKEY version 6.0 and will only run on hardware that will support CADKEY version 6.0.

4. The compiler for the source code is limited to the Metaware High C compiler and the GNU C Compiler from the Free Software Foundation because of the limitations of CADKEY version 6.0.

5. International Standards Organization/American National Standards Institute (ISO/ANSI) standards for dimensioning (Y14.5M) will be used.

# CHAPTER II

# REVIEW OF LITERATURE

## Application Development

With the increased necessity of unique software for different applications in industry today, companies are forced to create their own customized software applications in-house. With this type of development comes difficulties, such as, management and maintenance of the software (Baum, 1992). One company that was experiencing new software development was The Hartford, an ITT subsidiary. The Hartford wanted to create a high quality application software package at low cost and be able to maintain this software after it was in use. As this phase was implemented, it was found that development was taking too long and was costing too much. The solution for The Hartford was to install a new high level language to support COBOL, the existing language, then training for their programmers to submit good programs to the software library and finally the commitment of full-time resources to supporting the productivity of the programmers (Crawford, 1986).

Software development strategies can help combat problems and formalize methods for development with new software applications. This will also make the steps in getting new software developed easier and more succinct (Jones and

Shaw, 1990). Some of these strategies and methods include Computer-Aided Software Engineering (CASE), Software Engineering (SE), Rapid Application Development (RAD), and the Vienna Development Method (VDM). All of these strategies assist in the development of new software applications.

An area of development that has become somewhat of a standard practice is in the area of customization of existing software packages. These types of packages are called add-on programs (Madsen and Shumaker, 1993).

With most CAD systems, there are options that allow users to customize "off the shelf" packages to suit their own needs and to use enhancements created by other users (Kramer, 1993). Some of these add-on programs and options include Dynamic Link Libraries (DLL), Run-Time Linking (RTL), and shared objects (CADKEY, 1993). There are two other add-on programs that will be dealt with more in depth, AutoLISP for Autocad and CADKEY Dynamic Extensions (CDE) for CADKEY.

The AutoLISP programming language is derived from LISt Processing (LISP) which is the second oldest high-level language used by modern CAD systems (the oldest is FORTRAN) (Kramer, 1993). AutoLISP was designed by AutoDesk for use with Autocad. There are many advantages to using AutoLISP to customize Autocad such as, automatically creating shapes with text placed inside, drawing parallel lines by specifying beginning and end points, creating multiple lines of text in a preset style (Madsen and Shumaker, 1993), or control robot movements to make basic motions and to avoid obstacles (Kramer, 1993).

In addition to the previous add-on programs there exists another that may have more of an advantage over the others. The CADKEY Dynamic Extension (CDE) mechanism takes advantage of the powerful features of CADKEY and goes a step further by customizing ideas and programs created by the user. The CDE program contains the code and data for the given processes to be carried out within CADKEY. There are many CDE programs that exist within the standard package of CADKEY and with other third party application packages that can be purchased. These include Picture It, CADKEY Analysis, the on-line CADKEY tutorial, and the CADKEY Advanced Modeler.

Another example of a CDE program in use today is MF/Link created by Software Ventures, Inc. This program allows CADKEY part files to be transferred to Moldflow part files by the use of menu selections created by the CDE code (Software Ventures, Inc., 1992). This is a software package that was created by a third party company because of the need for this application by current CADKEY and Moldflow users and made possible by the capabilities of the CDE mechanism within CADKEY.

The CDE mechanism provides a function that will call back and forth between the CDE module and CADKEY. The CDE mechanism provides the necessary loading of the code into CADKEY and then establishes the connection for the call back between the two. One advantage of this process is that once the CDE module is loaded, it will run as if it is part of the standard CADKEY software (CADKEY, 1993).

There are many other advantages to the CDE module, which include: (a) a more productive channel for software development at CADKEY; (b) more power to the developers for their applications, since this is a modular program; (c) more versatile and responsive CADKEY programs for the user with more choices for changes that the user feels necessary; (d) characteristics within the C programming language; and (e) many other advantages over other add-on programs for application development (CADKEY, 1993). The advantages are to the users and the developers who are now able to create very large applications without loss of performance because of the modularity of this mechanism. There is also another advantage to the user because they are now able to pick and choose between the available additions and applications and then build a CADKEY with only the modules that are needed for their own unique applications.

There does exist one limitation to the CADKEY user when executing a CDE module. While the CDE mechanism is running, the user is not able to use the standard CADKEY functions accessible from the menus, such as, dimensioning or addition of necessary entities that are not included in the CDE program. But the user can exit the CDE module, perform the desired CADKEY function, and then enter the CDE module again. This task takes very little time and does not inhibit the user in any way.

Some of the experts in the CDE development field include Tecnocad Ltd. in Sligo, Ireland where Ken Carrol has developed CDE mechanisms for CADKEY for the plastics industry. Software Ventures, Inc. in Kalamazoo, Michigan has created a link between CADKEY and the Moldflow process simulation software for the plastics injection molding industry.

## Programming Languages

When developers of Computer-Aided Design systems decide to create add-on type of capabilities to their software they must choose a programming language in which the code is written for execution. When choosing a programming language there are many items to be considered. These include: (a) ease of translation from design to code, (b) efficiency of the compiled code, (c) portability of the source code, (d) availability of development tools, and (e) long term maintainability of the code. This step is more important than may be realized at the onset of the design process. Most software creators will tend to consider the support tools and how they will work together to more quickly implement the software rather than the type of programming language to use. This kind of decision may lead to sub-optimal performance and maintainability of the application software (Montgomery, 1991). (Unfortunately, the end user and third party developer do not usually choose the programming language but must accept the language that the supplier has chosen.)

There are many options that can be chosen for programming languages depending on the end use of the software. This discussion focuses on those associated with CAD software. All of the languages discussed are high-level languages because of their portability and ease of use.

## FORTRAN

FORTRAN is the most popular programming language for engineering and scientific work. FORTRAN is a second generation language (first generation is represented by Machine and Assembler languages). Second generation languages have the capabilities of huge software libraries and wide acceptance and familiarity (Montgomery, 1991). Also, FORTRAN is a good language for number crunching for analysis software applications (Lehmkuhl, 1983).

FORTRAN is a procedural approach to programming. A procedural language is one that each statement in the language tells the computer to do something and is basically a list of instructions. When these programs get very large, it gets very difficult to decipher the code so subroutines are created according to certain functions within the program to make it easier for the programmer (Lafore, 1991).

One disadvantage to FORTRAN is the method it uses to saves data. The manner in which the data is stored is critical and if the arrangement of data changes, all of the functions that call to this data in the program must change as well. This can be a very difficult task because many lines of code will have to be adjusted (Lafore, 1991).

## C Language

The C programming language has its roots in the development of the

UNIX operating system from AT&T Bell Laboratories. Even though this is why C was created, C is now used across many operating systems and most of the major personal computer software uses this language for creation of its applications. This language is used for creation of graphics packages, spread sheets, word processors, scientific and engineering applications, and CAD/CAM applications (Hutchison and Just, 1988). C has many powerful features which make it a more advantageous programming language. These features include complex data structures, extensive use of pointers, and many operators for computation and data manipulation (Montgomery, 1991). This project will be using the C programming language.

## Compiling and Linking

For every high level language, a compiler is needed to translate the program from man-readable to machine readable. This translation is necessary because the source file that is created by the programmer is in ASCII format and the computer reads files in binary format or machine language. The source file that is created by the programmer is not an executable file (Lafore, 1991).

To create an executable program there are two steps that must be accomplished successfully. These include compiling and linking. A compiled program contains machine-language instructions that can be executed by the computer. But these instructions are not complete. This compiled file is usually called an object file (Lafore, 1991).

Once the program is in machine language instructions (object file), the program must be linked to create an executable file. This file can then be executed or run by the programmer or end user to accomplish the intended task of the program.

In customizing existing CAD software, there arises some limitations when using a compiler. For CDE development in CADKEY, there are only three types of compilers that are accepted, Metaware High C compiler, the Watcom compiler and the GNU C compiler (GCC) from Free Software Foundation. These compilers will produce the object and executable files for the source program. The reason that a programmer must use this compiler is because CADKEY is a 32-bit application and all extensions used with it must be 32-bit applications as well. Since CADKEY was compiled using Metaware High C compiler, this is the compiler that is recommended for all applications used with CADKEY (CADKEY, 1993). The Watcom compiler is another 32-bit compiler. But the GNU C compiler, which can be downloaded from CADKEY's Bulletin Board, can also be used to develop CADKEY Dynamic Extension modules. Hopefully this limitation will change in the future, since this may inhibit the software developer and certain companies from using CDE modules for creating customized applications.

## Mold Design

Computers and computer-aided engineering (CAE) have become an integral part of the plastics industry. Since the onset of CAD, designers of plastics parts have had increased productivity when creating models and detail drawings (Forbes, 1989). Most part design done today is modeled in either two-dimensional or three-dimensional geometry on a CAD system.

For the injection molding industry, mold and part analysis have become a true advantage to all molders. Finite element analysis and process simulation have eliminated wasteful trial-and-error methods of the past and have given companies, who use these types of tools, an edge on their competition (Forbes, 1989).

Most of these analysis tools are targeted for part design. Programs, such as, Moldflow, C-Flow and Plastics and Computers have finite element methods for determining and analyzing flow front advancement, weld line placement, pressure to fill cavity, and air and gas trap locations (Vallens, 1993). Software has been developed for cooling analysis of the mold and part, as well. This type of software analyzes the placement of cooling lines and the effects they have on the temperatures in the part and the mold (Vallens, 1993).

These software packages were the first step in analyzing the mold instead of just the part. There is also analysis software that will determine the shrinkage rates across all critical dimensions of a molded part. This type of analysis drastically reduces lead times and costs for precision mold tooling, while preventing production delays caused by mold revision (Mold Design Software..., 1989). In other words, there are several software packages available to companies in the plastics injection molding industry for part analysis but there seems to be a shortage of software for those involved in injection mold design.

Computer-Aided Design has helped this shortage but usually does not handle dimensions or part features in ways that work well with the machining operations required to make injection molds. A few software packages do exist that aid the designer in the mold creation on a CAD system. One such package is Micro Cadam Plus from Altium. One advantage of this system is that it automatically draws a side view using the user-picked geometry from the plan view of the injection mold. Another advantage of the Micro Cadam Plus system is for Electric Discharge Machine (EDM) operations. It can compensate for the overburn created by the EDM machine as they cut the graphite tool (Mold Makers..., 1993). Unfortunately, this is not a true injection mold design system.

A software package was developed by Engineering Technology students at Western Michigan University named Design-A-Mold. This software uses CADKEY Advanced Design Language (CADL) for system code and can be executed with the CADKEY design system. The program creates a three-dimensional model of the injection mold and its standard components based on D-M-E standard catalog A and B series mold bases. From this three-dimensional mold model, layout drawings of the ejector system, cooling system, and other external systems needed to create the tool can be added with standard CADKEY operations (Branch, Brown and VanderKooi, 1992). This system represents a true mold design system since everything that comes with a standard mold base from D-M-E is drawn automatically by the Design-A-Mold software.

Another mold design system is R12/MOLD from AutoCAD. This software uses the AutoCAD dialogue boxes for dimensioning and detailing in AutoCAD much easier and faster. Features in this new software include ordinate dimensioning and automatic doglegging. A library containing both National and D-M-E mold bases are included in the software package (E D Sales and Service, 1994).

Other types of mold base design systems that exist or have existed in the past include: (a) Unisys' CAD/CAM system that will design and draw customized mold bases automatically, (b) MoldMate from Vector Automation contains all of the D-M-E mold bases as preset drawings that can be called up from inside this system, and (c) the CADKEY division of Micro Control Systems, Inc. developed an interface to D-M-E's mold base-building software that could create three-dimensional mold bases within CADKEY (Lodge, 1988).

There are a few others that have become more popular than the previous software packages discussed. These include Engineered Dynamics from National Dynamics in Chicago, Illinois and MoldMaker from Matra Datavision in Israel. The National software is an electronic library or catalog which allows designers to become more efficient when designing injection molds using Autocad as the base system (National, 1993).

## Dimensioning Standards and Practices

In order to complete any design that has been created on a CAD system, a two-dimensional detailed layout drawing usually must be created. This layout drawing is used to communicate effectively with the manufacturer that will produce the part. Therefore, it is very important to include all detailed information required to manufacture the part. A detailed drawing usually consists of an orthogonal layout of the part along with the dimensions necessary to convey the proper information and describe the part correctly. Which is why it is very important to include all of these dimensions on the layout drawing.

The American National Standards Institute (ANSI) and the International Standards Organization (ISO) have specific standards that are setup for detailing a drawing. These standards are documented in the ISO/ANSI manual in the Y series, especially Y14.5M, Dimensioning and Tolerancing for Engineering Drawings (ANSI, 1988). This section specifies the standards that are to be used for dimensioning and tolerancing a layout drawing. These standards include rules for measurement units, standards for dimensioning various shapes, and rules for specifying repetitive features within a drawing. This project will adhere to these standards for all dimensioning of the mold base layout drawings.

In addition to the ISO/ANSI standards, there are also industrial practices that are used for dimensioning mold base drawings. Unfortunately, there is not just one uniform practice for the entire mold design industry. Each company has its own practices for the way that the mold layout drawing should be dimensioned. For example, A-Tech Mold uses ordinate dimensions to describe the mold base. They use the center of the mold base as the datum feature and measure all dimensions from this datum (See Appendix A). A-Tech Mold views this type of dimensioning as an advantage since it takes up less space on the drawing (Brown, 1994).

Ordinate dimensioning seems to be the norm in industry, but there is still some variation within this aspect as well. Although there are differences in the way that industry dimensions mold base layout drawings, the features on the mold base layout which they dimension, are basically the same. For example, most designers will always dimension the overall length and width of the mold base, as well as, all the plate thicknesses with standard horizontal and vertical dimensions. Other common dimensions include the centerline locations of all pins and holes, labels and notes describing the diameter or radius of these holes and pins, the total stackup dimension for all of the plates, and centerline measurement of water lines from the center of the mold.

The combination of the ISO/ANSI standards for dimensioning layout drawings and industry practices will facilitate the dimensions used in this project and will be accepted by most mold designers in the injection molding industry.

# CHAPTER III

# METHODOLOGY

This project will entail the creation of an automatic dimensioning feature which can be added to the D-M-E Company's Ultimate® mold base design software. In order to complete this project, there were several steps that were completed. A gant chart is available in Appendix B which shows a time line of all the steps about to be explained.

## Preliminary Review

### Step One: Made Initial Contacts

Initial steps were made to contact the D-M-E Company and Tecnocad, Ltd. in order to begin procedures to acquire one copy of the Ultimate® mold base design software. Once this software had been sent and received by Western Michigan University, it was loaded on one machine within the Industrial and Manufacturing Engineering department. To become familiar with the Ultimate® mold base design system by the D-M-E Company, review of the software began. This step, review and analysis of the software, continued throughout much of the project.

### Step Two: CADKEY Information

Contact was made with CADKEY, Inc., since this is the parent Computer-Aided Design (CAD) software package that the Ultimate® software executes on. Initial contact was with Tom Landry, the Regional Manager for CADKEY, Inc. CADKEY, Inc. sent the necessary information and literature needed to start development in the use of CADKEY's Dynamic Extension (CDE) protocol. The CDE module enables the Ultimate® software to run in the CADKEY environment while utilizing the advantages of the standard CADKEY software (CADKEY, 1993).

## Compilers

### Step Three: Found Compilers

The necessary compilers were found to create the executable files once the C program has been created for the CDE module. There are only three compilers that can be used with CADKEY. These include the Metaware High C compiler, the Watcom compiler and the GNU C compiler (GCC) from the Free Software Foundation (FSF). The GNU C Compiler was ported from the Unix platform to a compiler for use on a 386 or 486 MS/DOS Personal Computer (PC) (CADKEY, 1993). The GNU C compiler was used first since it was readily available and it was free. It was decided to use this compiler until it was found necessary to purchase one of the others.

### Step Four: Setup GNU C Compiler

The GNU C compiler was downloaded from the CADKEY, Inc. bulletin board for use in the development of CDE programs for this project. This compiler was loaded onto a personal computer (PC) and the files were extracted from the original downloaded file and put into the correct directories on the PC per directions from CADKEY. Environment variables had to be added to the initialization files on the PC (autoexec.bat and config.sys) in order to execute the GCC in an organized fashion. Once this was done, the GNU C compiler was ready for utilization. It is at this point that development began on the C programs that would become CDE modules. It was also necessary to determine the differences between the three compilers. (This will be discussed in an upcoming step.)

## Development of Test Programs

### Step Five: Initial C Programs

Initial programs were created that were relatively simple in order to test the GCC and become familiar with the methods in creating a CDE module from a C program. The first program created showed text in the command line once it was executed in the CADKEY environment. This was a sample program that is shown in the Exploring CADKEY's Open Architecture manual obtained from CADKEY.

### Step Six: Graphical C Program

The next step was to create a program that will create graphics on the screen, such as lines and arcs. Once this step had been accomplished, a CDE module that created a rectangle on the screen and then dimensioned the length and width of this rectangle was developed and compiled.

## Background Information

### Step Seven: Research

While the development of these CDE programs were being performed, research continued to gather sufficient background information for this project. This research included information on C programming and other mold design software on the market.

### Step Eight: Formal Agreement

During this phase of the project, a proposed formal agreement was created between Tecnocad Ltd. and Western Michigan University regarding necessary information that was needed to complete this project. This type of information included time frames for answers between both parties, the license agreements regarding the software, time frames for completion of program, warranties and disclaimers, and use of the new feature in the original Ultimate® software after it was completed.

### Step Nine: D-M-E Competitors

Contact with D-M-E and Tecnocad Ltd. also continued on a need-to-know basis as the project progressed. Information regarding competitors of the Ultimate® mold base design software were researched and reviewed. The main competitor was National Tool since they have also created software which created mold bases from their catalog.

### Step Ten: Tecnocad Information

Constant contact with Tecnocad Ltd. was necessary for a more in-depth view of the software regarding the details of the programming language. Other information was also desired regarding their programming techniques which aided in the development of this new feature of the software.

## Standard Dimensions

### Step Eleven: Determine Standard Dimensions

One very important aspect of this project was the determination of the correct standard dimensions to use on the mold base layout that is created using the D-M-E Ultimate® software. This was vital because the mold layout drawing is really a communication tool to relay information to the mold maker from the customer. If this communication is not clear, there can be serious problems with the end result or in this case the mold itself (Lange, 1984). This can result in difficulties in relations between the two companies and possibly a loss of an account.

To combat these types of problems, it is important to make sure that all of the necessary information is on the mold design layout drawing. One way to make sure that this is accomplished is to make it almost impossible not to include all standard information on the mold base layout.

The determination of necessary dimensions and the placement of these dimensions was determined by sampling several mold design practices from both industry and training institutes. A survey of mold designers in the West Michigan area was conducted which aided in determining the necessary mold dimensions for proper communication and practice of mold layout drawings used today. Names of mold designers and injection mold design companies in the West Michigan area were contacted for permission to send out the survey to them. Those that accepted were sent a letter and a mold layout for them to dimension as they would any other mold layout produced by their company.

Another source of information was the area community colleges which train mold designers. Kalamazoo Valley Community College (KVCC) was contacted for training materials and techniques that are used in training mold designers. All of these sources were used to determine the data positions on the mold base layout for dimensioning purposes. The compilation of the data from the survey can be found in Appendix C.

### Step Twelve: ANSI/ISO Standards

Along with the survey of industry practices, International Standards Organization (ISO) and American National Standards Institute (ANSI) standards were researched so that the dimensions created will comply with these standards. The combination of ANSI/ISO standards and industrial practices ensures that the automatic dimensioning feature will be advantageous and useful to all who use the Ultimate® mold base design software and its added feature.

## Final Mold Base Program

### Step Thirteen: Variable Names

Detailed information was obtained from Tecnocad Ltd. concerning the locations on the mold base layout that were used for dimensioning positions. At this point, it was decided that an external data file would be created by Tecnocad Ltd. which outputs X and Y coordinates of the predetermined positions on the mold base. The new feature program would then read from this data file for all necessary information on dimension placement and positions.

### Step Fourteen: Mold Base CDE Module

Creation of CDE modules continued with the development of more complex programs for continued progression toward developing the automatic dimensioning feature. One such program took one mold layout created by the

Ultimate® software and created the determined dimensions by reading the X and Y coordinate points from an external ASCII data file.

The data file was created by creating a new mold base in the Ultimate software. Once this was displayed on the screen, the coordinates for the dimensioning positions were recorded in the data file. The program then retrieved the coordinates from the data file to be used to create dimensions on the mold base layout. This was the first programming step that incorporated a mold base in the actual CDE module.

### Step Fifteen: Met with Tecnocad Ltd.

At this time, the personnel from Tecnocad Ltd. were in the United States for a meeting with The D-M-E Company. While they were here, a meeting was scheduled for a discussion on the external data file and the ability of the Ultimate® software to generate the data file automatically. After this meeting, several problems were worked out with the format of the data file. And a new set of software was loaded that would generate the data file from within Ultimate® automatically.

### Step Sixteen: Final Program- Phase 1

Once successful compilation of the program with the sample mold base and dimensioning feature had been done and the new software had been loaded, the final program was started. This program will include the necessary information for reading from the external file created by Ultimate® and Tecnocad Limited. At this point, the external data file was only partially finished. It only incorporated the first two views of the four view mold layout. So the final programming for phase one was done so that it only incorporated the two views that were included in the data file.

### Step Seventeen: Final Program - Phase 2

Once the final version of the Ultimate® software was received and installed, the rest of the programming was finished. This updated version of Ultimate® is able the generate the data file with information from all views. This software was able to create a complete external data file so that the rest of the automatic dimensioning program could be finished.

### Step Eighteen: Testing

It was very important to test the capabilities of this new automatic dimensioning feature with many of the mold bases within the D-M-E software. This testing was very extensive but was necessary to make sure that all information was correct and displayed correctly.

The new feature program runs as a separate CDE module that must be run after the Ultimate® software has been used and a mold base saved to disk. This will not inhibit the testing of the module but it will be accessed differently once it has been added to the original Ultimate® software.

When the final program had been thoroughly tested, a copy of the final program was sent to Tecnocad Ltd. for possible addition to the original Ultimate® software program.

# CHAPTER IV

# SOFTWARE DESIGN AND DESCRIPTION

## Preliminary Review

In the early stages of this project, it was necessary to become acquainted with the companies and people that were going to be involved with this project. This included contact with CADKEY, Inc., the D-M-E Company, and Tecnocad Limited. CADKEY is the Computer-Aided Design (CAD) software package with which the Ultimate® software executes and D-M-E is the company that markets the Ultimate® software. Tecnocad Ltd. is the company who created the original Ultimate® software and gave the necessary information about the software to enable the creation of the automatic dimensioning feature for Ultimate®.

Within this phase of the project, it was necessary to obtain the vital information from CADKEY to create the CADKEY Dynamic Extensions (CDE) modules. Once these formalities were taken care of, the next steps were started.

## Compilers

At the beginning of the project, three compilers were found that could be used. However, it was unknown if any of them would actually work. The first two compilers were the Metaware High C compiler and the Watcom compiler which create 32-bit applications and are accepted by the CDE loader in CADKEY. The third compiler is the GNU C Compiler from the Free Software Foundation (CADKEY, 1993). This compiler was readily available from the CADKEY bulletin board. The GNU C compiler and the software development kit (SDK) for CDE development from CADKEY were downloaded from the CADKEY bulletin board. The SDK contains the necessary executables, header and object files for creating CDE's using the GNU C Compiler (CADKEY, 1993).

The first two compilers cost over one thousand dollars each. So it was decided to start with the GNU C Compiler, since the only cost was the phone connection to CADKEY's bulletin board. After the final program had been completed, it was found that the GNU C Compiler worked adequately for creating all CDE files to run the new feature and complete the project and it was not necessary to purchase either of the other two compilers.

## Development of Test Programs

In order to gain a good understanding of all of the characteristics of programming using the CADKEY Dynamic Extension (CDE) module and C programming in general, several programs were created. These programs ranged from very simple text output to more complex graphics and dimensions displayed in the CADKEY environment. The following is a list of several of the programs that were written for the express purpose of learning the different attributes necessary to complete this project.

## Text Programs

The first programs that were created were relatively simple but were necessary to test the capabilities and format of the CDE module. The first step in creating any C program is to write the program using a text editor, such as Word Perfect or Microsoft Word. But when using these types of editors, it is necessary to save the file in ASCII format so that the compiler is able to read the program. Since the C programming language is a high level language, the ASCII text program that was produced in the text editor must be comiled from a man-readable file to a machine-readable file. A compiled program contains machine-language instructions that can be executed by the computer but these instructions are not complete (Lafore, 1991).

Once the program is in machine language instructions (or object file), the program must be linked to the correct libraries to create an executable file that can be run by the user. These libraries contain the necessary information for the computer to understand the types of functions that were used in the program. Once the file has been linked successfully, it can be executed or run by the programmer or end user to accomplish the intended task of the program. For every program that was created for this project, these steps (editing, compiling, and linking) must be executed in this order.

The first program that was successfully developed simply printed text in the prompt area of the CADKEY screen (See Appendix D, page 65). This program was created to test the capabilities of the compiler and linker. And to check the ability to create a CDE module that would execute successfully in CADKEY.

### Graphics Programs

The next programs included graphics as well as text. These programs drew lines to user specified locations and to predetermined locations set within the program (See Appendix D, page 66). All of the graphics and text were displayed in the CADKEY environment.

The next interactive graphic program was able to draw a rectangle by user defined positions for the corner points (See Appendix D, page 68). A progression into other areas for graphics development was explored for further understanding of CDE characteristics. This included creation of circles and arcs. This was done with user entered information for placement in the CADKEY environment. Once these graphics programs were executed successfully, the next step was to create programs that dimensioned the graphics that had already been created.

### Dimensioning Programs

The first dimension program that was created just used standard horizontal dimensions which were dependant on user input for the text position (Appendix D, page 70). The dimension programs were added to the graphical programs so that once the object, such as a rectangle, was created it could then be dimensioned

(See Figure 1).

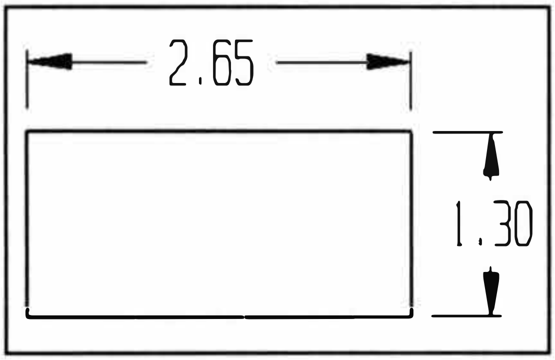


Figure 1. Dimensioned Rectangle.

The dimensions created were dependent on the size of the rectangle that the user entered and would change as the rectangle changed size (See Appendix D, page 72). These programs only used standard overall horizontal and vertical dimensions.

The next program sample developed had the capability of creating a rectangle, with user entered corner positions, and a user positioned circle. After these graphics were created, the program dimensioned these two objects (See Figure 2).

The user had to choose the position of the dimension text on the CADKEY screen with the cursor (See Appendix D, page 74). All these programs ran continuously, or in a loop, until the user pressed the RETURN key on the keyboard. This was done so that the program could run several times to test the integrity of the program.

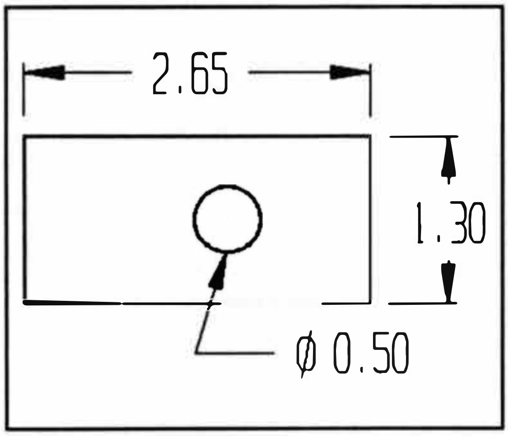


Figure 2. Rectangle and Circle Graphics With Dimensions.

Once both graphics and horizontal and vertical dimensions could be displayed successfully, other dimensioning programs were created. The survey of mold designers showed that many used ordinate dimensioning for detailing their mold layouts. (Refer to Figure 3 and 4 for differences between standard dimensioning and ordinate dimensioning.) The next programs that were created incorporated ordinate dimensioning to reflect the type of dimensioning used in the results of the survey. The ordinate dimensioning program used both vertical and horizontal ordinate dimensioning (See Figure 4 and Appendix D, page 77).

Within the ordinate dimensioning option in CADKEY, there were several areas had to be examined to make sure that it was understood how this function operated. These areas included positioning of relative points from the base position and editing an ordinate dimension once it was created.

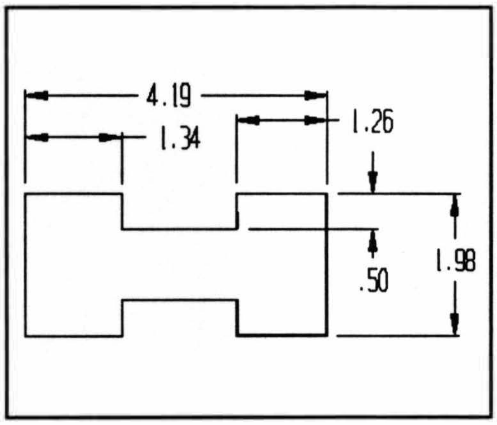


Figure 3. Standard Dimensioning.

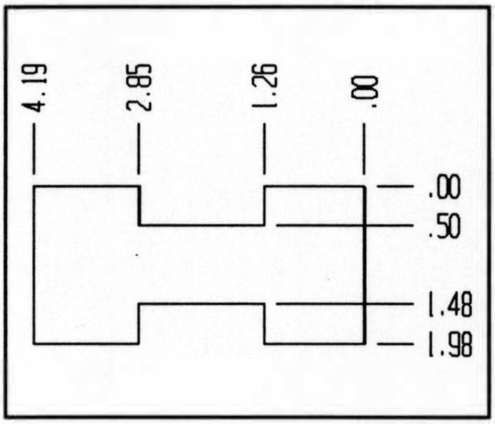


Figure 4. Ordinate Dimensioning.

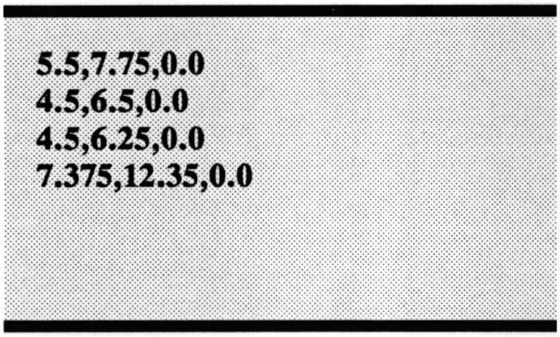
Once these obstacles were overcome, other areas had to be explored before the final program could be started.

External File Programs

After talking with Tecnocad Ltd., regarding the final program, it was decided that the automatic dimensioning CDE module would read from an external file for the X and Y coordinates for dimensioning position on the mold layout. The information in the file included all positions for dimensioning the mold layout, including return pin locations, stop pin positions, and the clamp slot location (Refer to Appendix E for a mold layout drawing with these components included).

At this point it was vital that a program be written to pull the coordinates from the external file into a program that could use this information for dimensioning. Several programs were created to open and extract these data points from the file (See Appendix D, page 80).

One program was developed to output this one data line from the external data file to the CADKEY prompt line as text. This step was necessary because the coordinates that were read from the data file were retrieved as one line of text (a string character), even though it was actually three distinct positions (Refer to Figure 5).



```
5.5,7.75,0.0
4.5,6.5,0.0
4.5,6.25,0.0
7.375,12.35,0.0
```

**The external file consisted of four lines of data with three coordinate positions in each line. Each line was read in as one character (5.5,7.75,0.0) and then divided (unpacked) into seperate variables ([5.5], [7.75], [0.0]).**

Figure 5. Format of External Data File.

After this program was completed, a function was needed to break the text line into separate entities to retrieve the X, Y and Z coordinates of the dimension position. This was accomplished with the *sscanf* C function (See Appendix D, page 81).

Once these initial programs were created, the actual external data file format was given by Tecnocad Limited. This file format was not the same as had been used in the previous programs that retrieved information from the data file.

Because of this difference, more programs had to be developed to test the functions necessary to retrieve this new format. In this new data file, there was exrtraneous information provided that was not necessary for the automatic dimensioning function but was necessary for other functions within the Ultimate software. Only the X and Y coordinate positions were needed. An example of the new data file format is shown in Appendix F. After this obstacle was overcome, the final program was started.

## Background Information

Throughout the project, continued research was done to collect information in several areas, including application of functions within the programming language of C, continued review of the Ultimate® software, and the development of the formal agreement between Western Michigan University and Tecnocad Limited. Also more information was obtained from Tecnocad Ltd. concerning the type of functions that were used within the original Ultimate® software, such as, sorting functions for the X and Y coordinate positions, which were necessary for the automatic dimensioning feature.

## Standard Dimensions

Before the final program was started, the standard dimensions for the automatic dimensioning feature within the Ultimate® software needed to be determined. This determination started by contacting the D-M-E Company and asking what their recommendations for this feature would include. The only suggestion that they had was to follow the dimensioning format of the D-M-E catalog that shows layouts for most mold bases that can be purchased through D-M-E (See Appendix E for an example of the D-M-E layout).

After reviewing the dimensioning format of the D-M-E catalog, it was concluded that more research needed to be done to identify the practices of mold designers in industry today. To accomplish this, a list of mold designers and plastics engineers in the West Michigan area and other possible sources in the United States was developed. The names of individuals, companies, and professional societies came from the thesis committee members and other industry reference materials, which included the Plastics Encyclopedia and a listing of plastics industry professionals produced by the Society of Plastics Engineers.

The first step in determining the required dimensions and their locations was to contact the Society of the Plastics Industry (SPI) and the National Tooling and Machining Association (NTMA) for possible publications on detailing and dimensioning standards of mold base design layout. The Introduction to Mold Making by E.L. Buckleitner and Moldmaking and Die Cast Dies for Apprentice Training by John Kluz were two books recommended by the NTMA. Both books were acquired and reviewed. Unfortunately, neither book provided standard practices in dimensioning within the injection mold making industry.

From all industry contacts that were made, there did not appear to be any written documentation on detailing and dimensioning mold base layouts. Because of this lack of formal standards, it was necessary to conduct a survey of mold designers in the plastics industry to determine industry practices on mold layout dimensioning.

The first step in this survey was to make personal contacts with mold designers and engineers that were directly involved with Western Michigan University. This was done to obtain individual detailing specifications used within those companies or to get direction in finding information on detailing mold layout drawings .

After contacting several companies and individuals, it was found that formal specifications were not documented within many companies or in any possible reference materials. One of the companies didn't even produce paper copies of the mold designs; instead, everything was kept in an electronic database for each project. Although this portion of the survey was not successful in obtaining written documentation on dimensioning mold bases, several other contacts were obtained from these companies for possible reference materials.

The next step in determining dimensioning practices was to contact mold shops in the West Michigan area for assistance. Thirteen plastics companies were contacted and seven of them agreed to participate in the survey. It was decided that a hard copy of a sample mold base that had no dimensioning or detailing would be sent to these participants . The mold base was a standard D-M-E mold base created by the Ultimate® software (See Appendix G for layout). A letter was sent out with the mold layout drawing to the participants (See Appendix H). The letter requested an example of a mold base layout produced by these companies be returned with the survey. Five of the surveys were returned for evaluation including samples of company mold layouts.

Another source for the survey was current Ultimate® users. Four Ultimate® users' names were received from the D-M-E Company's support staff. These users were contacted and three agreed to participate in the survey. All agreed that they would rather send samples of mold base layouts with dimensions from their company instead of the previous mold base that was used in the survey of West Michigan mold designers.

The most usable information on dimensioning a mold base layout was acquired from the actual mold designers that participated in the survey and sent back the required information. It was this information that was reviewed and applied to aid in producing an automatic dimensioning feature that would follow general industry practices that were defined and described by the survey. The results and details of the survey are shown in Appendix C. All of the dimensions that were used in the automatic dimensioning CDE module were determined by the results of the survey of mold designers in industry today.

From the survey, it was found that when dimensioning the top view of the mold base, also known as the B Plan View, both overall horizontal and vertical dimensions were used, as well as, ordinate dimensioning for specific locations of holes and components. The base ordinate position was the center of the mold (See Appendix G, page 87). From all of the surveys collected, while focusing on the top view, all but one participant used ordinate dimensioning and all but three also used overall horizontal and vertical dimensions. So it was determined to incorporate both types of dimensioning into the new automatic dimensioning feature for the top view. All of the locations that were dimensioned on the mold base were similar from one survey to the next. These locations included the overall width and height of the mold base and the center positions of the components.

The next view is the cavity layout, also called the A Plan View. After reviewing the survey, it was found that there was not a clear distinction on the type of dimensioning used. Four of the eight surveys used standard horizontal and vertical dimensions, while five of the eight used ordinate dimensioning. Three of the surveys used both types of dimensioning. The ordinate dimensioning was the majority type but upon further investigation into the surveys that used this type of dimensioning, it was found that the components that were used with ordinate dimensioning had already been dimensioned in the top view (B Plan View). It was determined that it was not necessary to dimension these components again. The dimensions that were necessary were the standard overall dimensions and the clamp slot location (See the A-Plan View in Appendix G, page 87).

The next standard view of the mold base was the section view, also called the Stack-up Length. In this view, standard overall dimensions for the height and width were used on the mold layout from the results of the survey. Also some of the survey participants dimensioned the thicknesses of the different plates of the mold in this view. Some participants dimensioned the thickness of the plates in this view and the End View. Other components dimensioned in this view were the locator ring, the sprue bushing radius and the sprue diameter (Refer to Appendix G for Ultimate® mold layout). These were all incorporated into the final automatic dimensioning feature.

The last view on the mold base layout is the end view or the Stack-up Width. After reviewing the results of the survey, six of the eight surveys used overall dimensions to describe the mold base. Five of the eight surveys also dimensioned the plate thicknesses and the width of the ejector bar and the pin retainer plate in this view. Other components dimensioned in this view are the locator ring, sprue bushing, and clamp slot width. Since all of these types of dimensioning techniques are necessary to describe the mold accurately, both were used with the automatic dimensioning feature.

The other aspect of determining the necessary dimensions was to review the International Standards Organization (ISO) and the American National Standards Institute (ANSI) dimensioning standards for engineering layout drawings. These standards are not specifically set up for injection mold design layouts but for any dimensioned/detailed engineering drawing layout produced today. The specific ISO/ANSI standard that was adhered to was entitled Dimensioning and Tolerancing for Engineering Drawings, section Y14.5M. This standard was reviewed and adhered to throughout the project.

## Final Mold Base Program

The final program must read the data from the external data file and use this data to dimension any mold base created by the Ultimate® software. To accomplish this goal, it was necessary to get the correct information from the file. Because the file contained more information than was needed, functions were created to read in just the X and Y locations of the features on the mold from the external data file.

Within this data file, there were four sections that related to the four views generated by the Ultimate® software. They include the top view (B Plan View), the cavity layout (A Plan View), the section view (Stack-Up Length) and the end view (Stack-Up Width). The final program was also divided into four sections which corresponded to the four views in the data file. Each view was a unique view with unique dimensioning types as explained in the previous section (See Appendix G page 87). Some views had standard horizontal and vertical dimensions, others had ordinate dimensioning, while others had a combination of all types.

The major challenge was to retrieve the correct data in the correct format. Once the X and Y positions were extracted from the file, it had to be determined which components they were describing. This was accomplished through the use of decision statements in the program. Once the decision was made, the correct dimension types were used to output the dimension to the CADKEY screen. This was done for each view of the mold layout (Refer to Appendix I for a copy of the final program).

All of the dimensions created through the automatic dimensioning feature will be on a separate CADKEY layer and in a different color so that they are easily hidden or edited by the Ultimate® user. Also, all dimensions will have three decimal places to the right of the zero for accuracy.

A new updated version of the Ultimate® software was obtained from Tecnocad Limited. This included the option to automatically output the dimensioning file for the injection mold created in the Ultimate® software. This option is available from the main menu in Ultimate®. The dimensioning data file will only be created if the user chooses this option within Ultimate®.

An environment variable can be set up on the computer running Ultimate® to specify the path name of the directory in which to store the dimension data file. This file will have a filename that is the same as the part file saved by the user within CADKEY with an extension of .CDO (CAMold (Ultimate®) Dimensioning Output).

Once the external data file feature had been added to the CAMold software, the testing of the automatic dimensioning feature was done. Testing was done by executing the automatic dimensioning CDE function after the .CDO file had been created by CAMold which is run using the CADKEY software (See Figure 6).

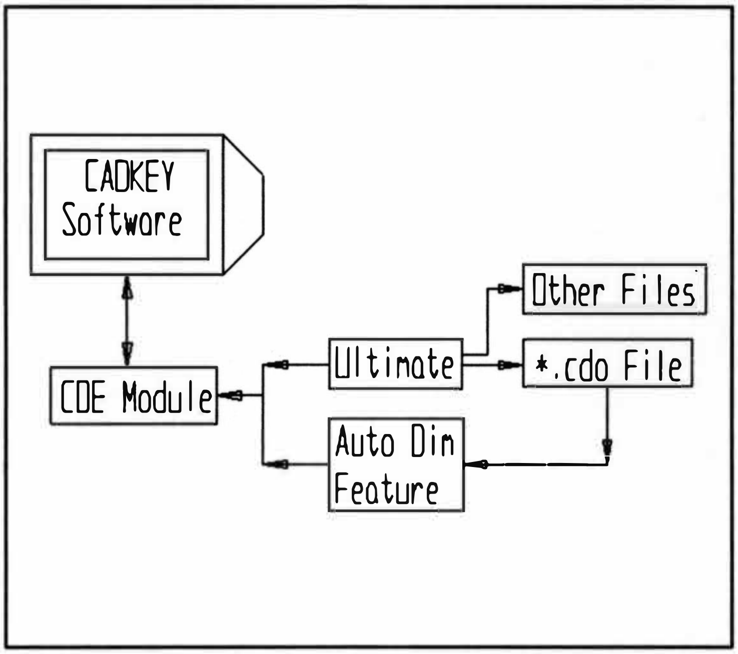


Figure 6. Schematic of Testing Procedures.

Many mold bases were selected and the dimensioning output files were created. All of the mold base layout drawings were checked for any errors or display problems. An example of a fully detailed mold base layout after the automatic dimensioning feature had been executed is shown in Appendix J. There were some minor changes made to the CDE file regarding dimensioning text position. Once these changes were made and tested, the program was finalized and sent to Tecnocad Limited for possible addition to the original Ultimate mold base design software.

# CHAPTER V

# CONCLUSIONS AND RECOMMENDATIONS

## Conclusions

The purpose of this project was to create an automatic dimensioning feature for plastic injection mold base design software. The new feature runs in conjunction with the Ultimate® mold design software from D-M-E. The automatic dimensioning feature was created to aid the mold designer in cutting tedious detailing and dimensioning work.

The following conclusions and observations were made based upon the data and information received throughout the project:

1. After completing the automatic dimensioning program, it was concluded that the GNU C compiler from the Free Software Foundation was adequate for creation of the CDE modules necessary to complete this project.

2. It was found, after researching competitors of the Ultimate® software, that no other mold design software incorporated an automatic dimensioning feature.

3. After conducting the survey of mold designers in industry, dimenioning practices were found for detailing and dimensioning a standard D-M-E mold base. These results can be found in Appendix C.

4. After receiving the final Ultimate® installation software from Tecnocad Limited, it was found that the automatic dimensioning output option did not output all of the data determined necessary from the survey. The data file that is output does not contain the necessary data positions for the Stack-Up Length view, as well as some other data positions for the locator ring, sprue bushing and the clamp slot width. This project was completed by creating a data file that followed the original format from Tecnocad Limited (Refer to Appendix F for the data file format).

This data file was used to assist in creating the dimensions for the automatic dimensioning feature used with the Ultimate® software. The only problem that this creates is in testing several of the mold bases with the automatic dimensioning feature. But the original data format provided by Tecnocad Limited has been followed and when the automatic dimensioning ouput feature has been corrected, further testing may be performed.

## Recommendations

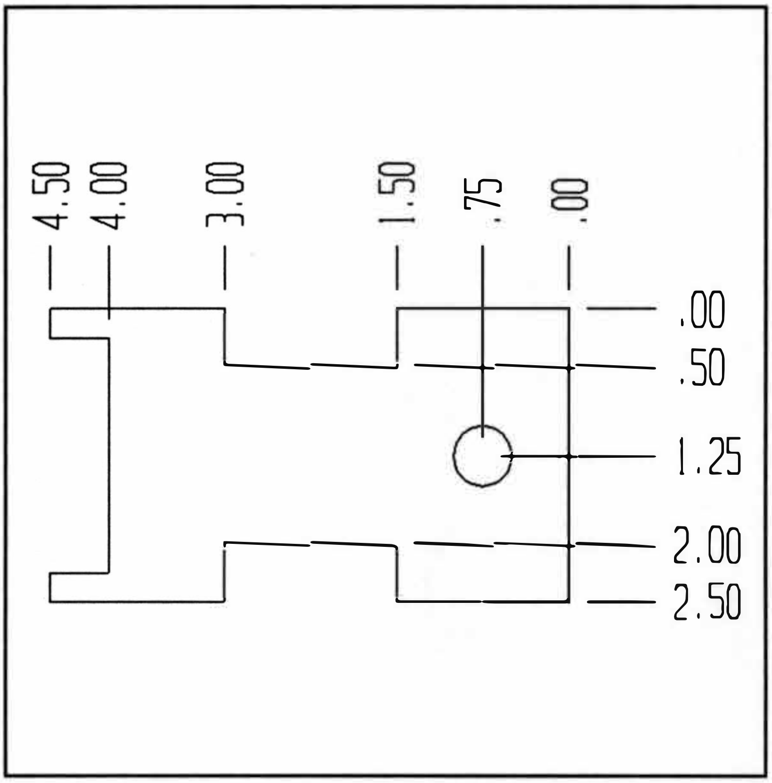
Since there were some difficulties in getting a working copy of the updated Ultimate® software, it is recommended that this be obtained before any further work or changes be made to the new feature. It is suggested that all mold bases be tested using this new updated Ultimate® software and the new automatic dimensioning feature since it was impossible for it to be done during this project. It is also suggested to get a copy of the Ultimate® software with the automatic dimensioning feature already incorporated into the software so that a complete test can be conducted to test the new feature.

It is also recommended that a more comprehensive survey of national and international mold designers be completed to confirm the dimensioning styles used in this project.

It is also suggested that background research be undertaken to determine if there are other new mold design software packages with an automatic dimensioning feature incorporated into them. This may lead to other possible changes and updates in the new feature created for this project.

# Appendix A

## Ordinate Dimensioning Example

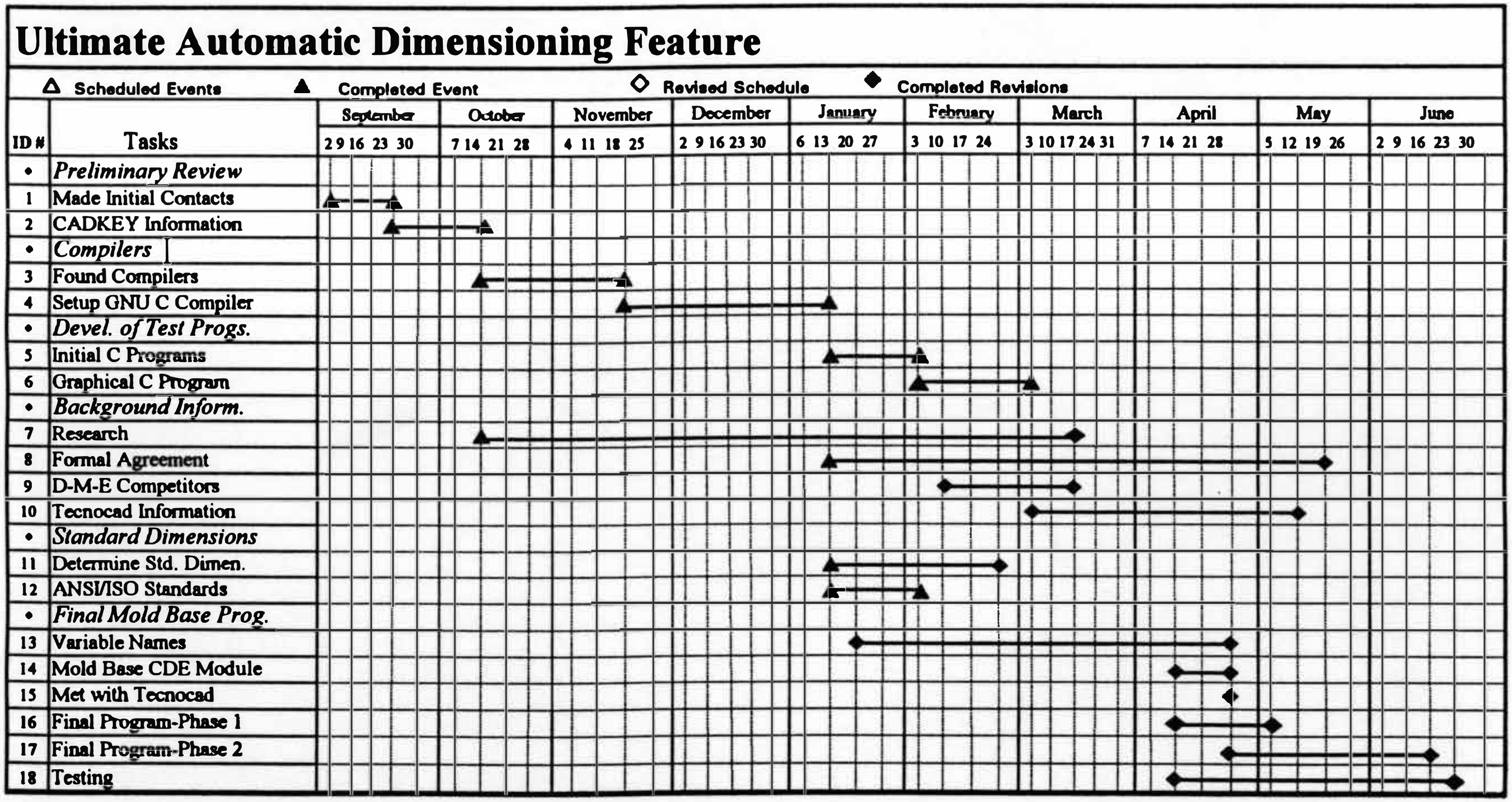


Ordinate Dimensioning Example

# Appendix B

## Gant Chart

# Appendix B - 1994/1995 Gant Schedule



# Appendix C

## Tabulated Information From Survey

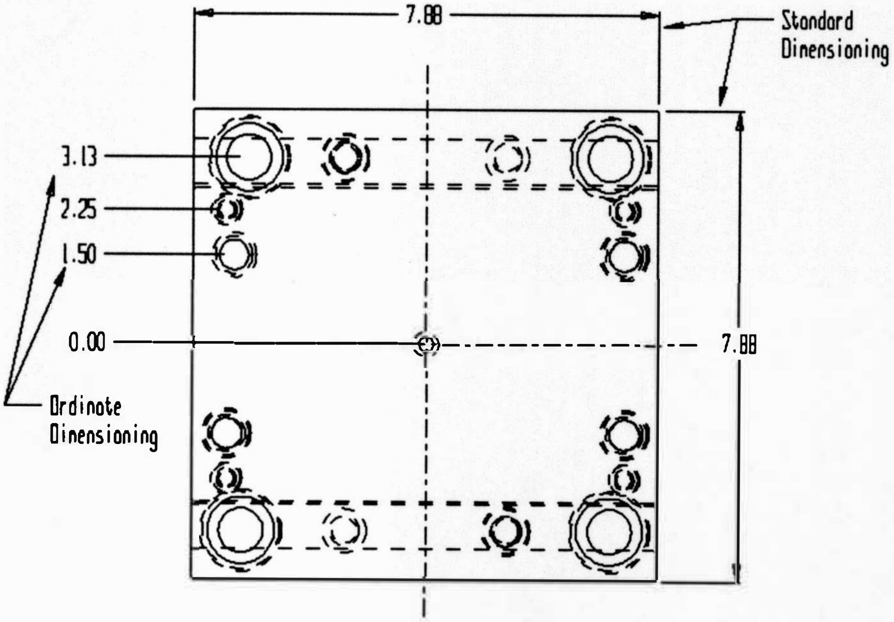
## TABULATED INFORMATION FROM SURVEY

| Example Number | | B Plan View | | A Plan View | | Stack-Up Length | | Stack-Up Width | |
|---|---|---|---|---|---|---|---|---|---|
| NUM | REL | STD | ORD | STD | ORD | OVR | THICK | OVR | THICK |
| 1 | 1A | | X | | X | X | X | X | X |
| 2 | 1B | X | X | X | X | X | X | X | X |
| 3 | 2A | X | X | | | X | X | X | X |
| 4 | 3A | X | | X | | X | | X | X |
| 5 | 4A | X | X | | | | | X | X |
| 6 | 5A | | X | | X | X | X | | |
| 7 | 5B | X | X | X | X | X | X | | |
| 8 | 6A | | X | X | X | X | X | X | |

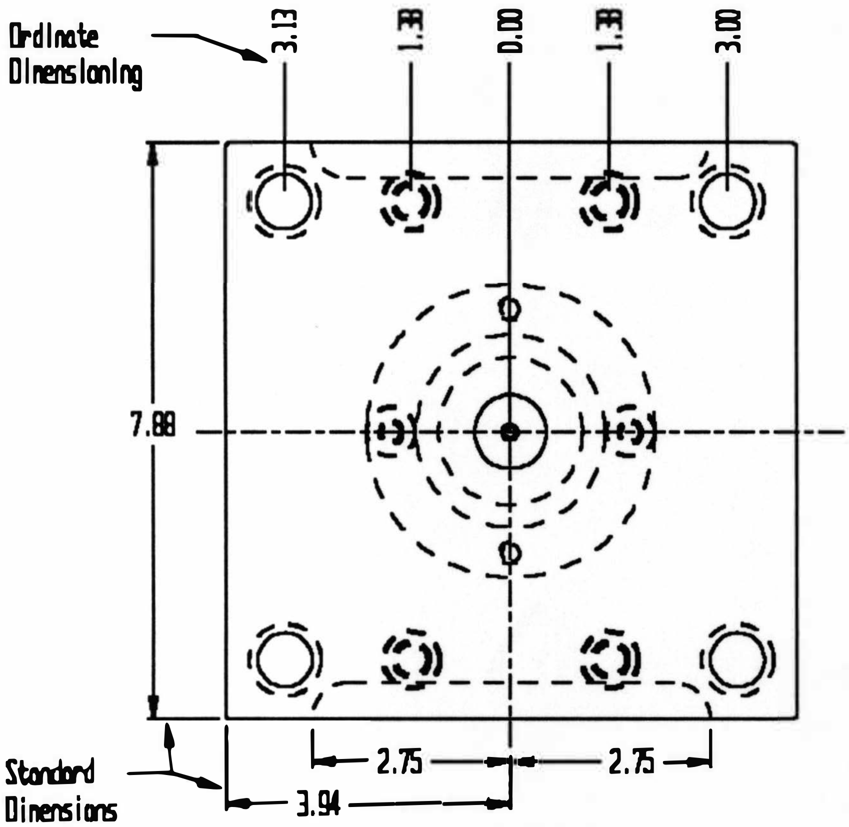
** Refer to the following mold layout for a graphic view of the information contained in this appendix

**LEGEND:**

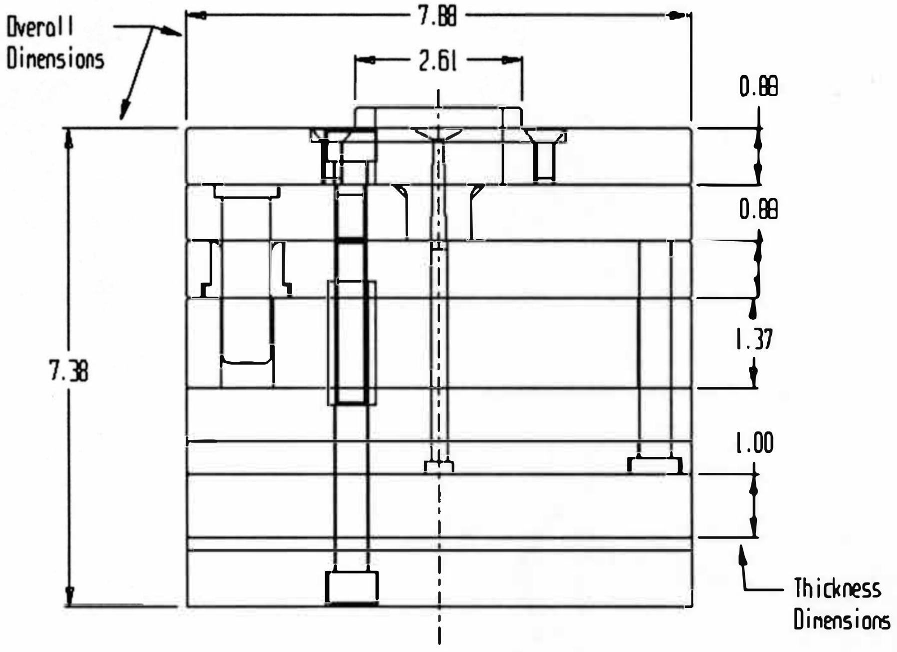
| | |
|---|---|
| **NUM :** | The number of the survey tabulated |
| **REL:** | The relationship of the surveys. Some companies sent more than one survey or example back.<br>For example, company number 5 sent back two examples, A and B. |
| **STD:** | Standard horizontal and vertical dimensions |
| **ORD:** | Horizontal and vertical ordinate dimensions |
| **OVR:** | Overall width, height, and length dimensions |
| **THICK:** | Dimensions for thickness of all plates |


7.88
Standard
Dimensioning
3.13
2.25
1.50
0.00
7.88
Ordinate
Dimensioning

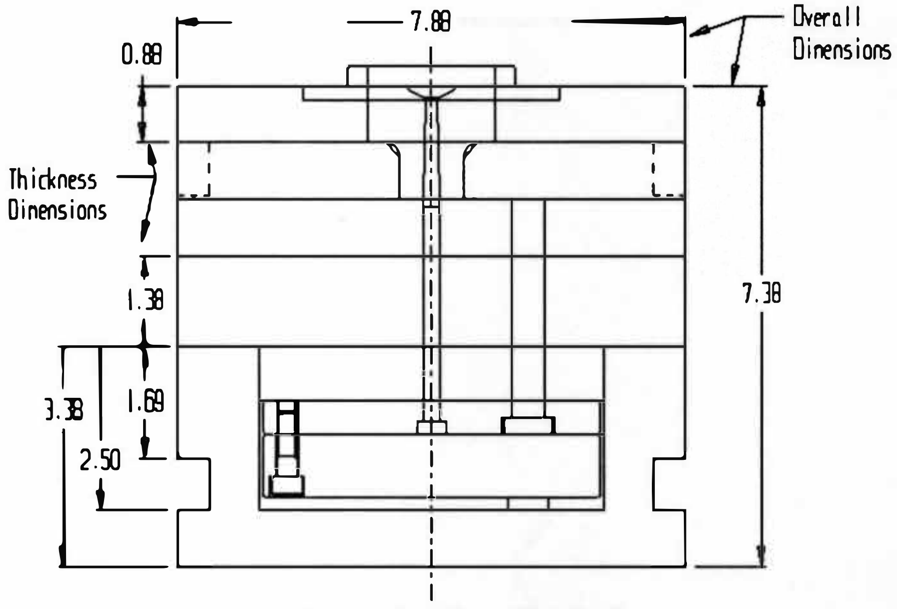
B Plan View


Ordinate Dimensioning
3.13
1.38
0.00
1.38
3.00
7.88
2.75
2.75
3.94
Standard Dimensions

A Plan View



Stack-Up Length



Stack-Up Width

# Appendix D

# Initial Programs

## Appendix D: Examples of Initial Programs - Text

Description: This program was created to test the capabilities of the CDE module. It will output the text "Hello, world!" to the CADKEY prompt line.

---

```
#include "/gccsdk/include/ck_cdl.h"

void hello()
{
  ck_pause ("Hello, world!");
}
```

## Appendix D: Examples of Initial Programs - Line

Description: This program was created to test the CDE module for creating graphics. It asks the user to indicate two positions on the screen for the start and end point of the line. It will output to the CADKEY screen one line using the default attributes set by CADKEY.

---

```
#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"

int
ln_plus ()
{
   int opt, stat, dblevel;
   double x1, y1, z1, x2, y2, z2;
   CK_ENTATT att;

   /* Get start and end positions for line */

   opt = 1;

getl:
   do {
      stat = ck_getpos ("Indicate start point", opt);
   } while (stat == CK_RETURN);

   if ((stat == CK_BACKUP) || (stat == CK_ESCAPE)) {
        return (0);
   }
   opt = stat;

   ck_getvar ("@xworld", &x1);
   ck_getvar ("@yworld", &y1);
   ck_getvar ("@zworld", &z1);

   do {
       stat = ck_getpos ("Indicate end point", opt);
   } while (stat == CK_RETURN);

   if (stat == CK_ESCAPE) {
        return (0);
```

```
    }
    if (stat == CK_BACKUP) {
       goto get1;
    }
    opt = stat;

    ck_getvar ("@xworld", &x2);
    ck_getvar ("@yworld", &y2);
    ck_getvar ("@zworld", &z2);

    ck_getvar ("@COLOR", &att.color);
    ck_getvar ("@LEVEL", &att.level);
    ck_getvar ("@LTYPE", &att.ltype);
    att.ltype = att.ltype + 1;
    att.grpnum = 0;
    att.subgrp = 0;
    ck_getvar ("@PEN", &att.pen);
    ck_getvar ("@LWIDTH", &att.lwidth);

    dblevel = 0;

    if (ck_getvar ("DEBUG", &dblevel) == CK_BAD_VAR) {
         ck_mkvar ("DEBUG", CK_V_INT, 0, NULL);
    }

    if (dblevel == 1) {
         printf ("LINE %f, %f, %f, %f, %f, %f\n", x1, y1, z1, x2, y2,
          z2);
    }
    else if (dblevel == 2) {
       printf ("LINE %f, %f, %f, %f, %f, %f, %d, %d, %d, %d, %d,
       %d, %d\n",x1, y1, z1, x2, y2, z2, att.color, att.level,
       att.ltype,att.grpnum, att.subgrp, att.pen, att.lwidth);
    }

    ck_line (x1, y1, z1, x2, y2, z2, &att);

    goto get1;
}
```

## Appendix D: Examples of Initial Programs - Rectangle

Description: This program was created to further test the graphic capabilities of the CDE module. It will ask the user for the bottom left-hand corner of a rectangle and the top right-hand corner of a rectangle. The program will take these points and create a rectangle in the CADKEY environment using default attributes.

---

```
#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"

int
rectangle ()
{
   int opt, stat, dblevel;
   double x1, y1, z1, x2, y2, z2;
   CK_ENTATT att;

   /* Get corner positions rectangle */

   opt = 1;

getl:
   do {
      stat = ck_getpos ("Indicate bottom left corner", opt);
   } while (stat == CK_RETURN);

   if ((stat == CK_BACKUP) || (stat == CK_ESCAPE)) {
      return (0);
   }
   opt = stat;

   ck_getvar ("@xworld", &x1);
   ck_getvar ("@yworld", &y1);
   ck_getvar ("@zworld", &z1);

   do {
    stat = ck_getpos ("Indicate upper right corner", opt);
   } while (stat == CK_RETURN);

   if (stat == CK_ESCAPE) {
      return (0);
    }
    if (stat == CK_BACKUP) {
        goto get1;
    }
    opt = stat;

    ck_getvar ("@xworld", &x2);
    ck_getvar ("@yworld", &y2);
    ck_getvar ("@zworld", &z2);

    ck_getvar ("@COLOR", &att.color);
    ck_getvar ("@LEVEL", &att.level);
    ck_getvar ("@LTYPE", &att.ltype);
    att.ltype = att.ltype + 1;
    att.grpnum = 0;
    att.subgrp = 0;
    ck_getvar ("@PEN", &att.pen);
    ck_getvar ("@LWIDTH", &att.lwidth);
    dblevel = 0;
    if (ck_getvar ("DEBUG", &dblevel) == CK_BAD_VAR) {
     ck_mkvar ("DEBUG", CK_V_INT, 0, NULL);
    }
    if (dblevel == 1) {
        printf ("LINE %f, %f, %f, %f, %f, %f\n", x1, y1, z1, x2, y2,
     z2);
    }
    else if (dblevel == 2) {
        printf ("LINE %f, %f, %f, %f, %f, %f, %d, %d, %d, %d, %d,
        %d, %d\n",x1, y1, z1, x2, y2, z2, att.color, att.level,
        att.ltype,att.grpnum, att.subgrp, att.pen, att.lwidth);
    }

    ck_line (x1, y1, z1, x1+(x2-x1), y1, z1, &att);
    ck_line (x1+(x2-x1), y1, z1, x2, y2, z2, &att);
    ck_line (x2, y2, z2, x1, y1+(y2-y1), z2, &att);
    ck_line (x1, y1+(y2-y1), z2, x1, y1, z1, &att);

    goto get1;
}
```

## Appendix D: Examples of Initial Programs - Dimensions

Description: This program was created to test the capabilities of the dimensioning features within the CDE module. It asks the user to enter the first and second positions that the horizontal dimension will cover. Then it asks the user for the text position. Once these questions have been answered, it will create the dimension in the CADKEY environment using the positions defined by the user.

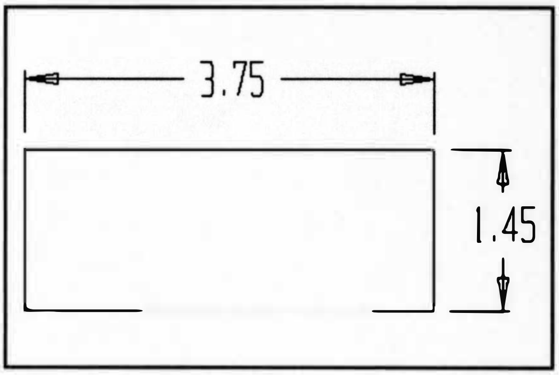
```
/* lindim.c */

#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"
void
lindim ()
{
   double pos[3];
   unsigned long id;
   int  vpno, stat, defopt = 3;
   CK_REFLN refln;
   CK_LOCDEF locdef[2];
   CK_ENTATT att = { CK_LNG_DFLT, CK_INT_DFLT, CK_INT_DFLT,
    CK_INT_DFLT,CK_INT_DFLT, CK_INT_DFLT, CK_INT_DFLT, 0, 0 };

get_pos_defl:

   do {
    stat = ck_getpos_def ("Indicate 1st position for HORZ dimension", defopt,
       &locdef[0]);
   } while (stat == CK_RETURN);

   if (stat == CK_ESCAPE || stat == CK_BACKUP) {
      return;
   }

   ck_getcoord (NULL, pos, NULL, NULL);
   refln.x1 = pos[0];
   refln.y1 = pos[1];
   defopt = stat;

   do {
    stat = ck_getpos_def ("Indicate 2nd position for HORZ dimension", defopt,
            &locdef[1]);
    } while (stat == CK_RETURN);

    if (stat == CK_ESCAPE || stat == CK_BACKUP) {
        return;
    }

    ck_getcoord (NULL, pos, NULL, NULL);
    refln.x2 = pos[0];
    refln.y2 = pos[1];
    defopt = stat;

    do {
     stat = ck_getpos ("Indicate text position for HORZ dimension", 1);

    } while (stat == CK_RETURN);

    if (stat == CK_ESCAPE || stat == CK_BACKUP) {
        return;
    }

    ck_inquire (CK_INQ_CURVP, &vpno);
    ck_inquire (CK_SET_VIEW, &att.vnum, vpno);
    ck_getcoord (NULL, pos, NULL, NULL);

    ck_lindim (pos[0], pos[1], 0.0, NULL, &refln, 0.0, CK_LDM_HORZ,
            NULL,NULL, NULL, &att);

    ck_inquire (CK_INQ_LASTID, &id);
    ck_set_locdef (id, 2, locdef);
    goto get_pos_def1;
}
```

## Appendix D: Example of Rectangle with Standard Dimensions

Description: This program used the rectangle that was created previously, and added both horizontal and vertical dimensions to it. The program asks the user for the horizontal and vertical positions for the text position.

---

```
#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"

int
dimension_1 ()
{
   double pos[3];
   int opt, stat, dblevel;
   double x1, y1, z1, x2, y2, z2;
   CK_ENTATT att;
   CK_REFLN refln;
   CK_REFARC refarc;
   double x,y;
```



**Output of program shown above.**

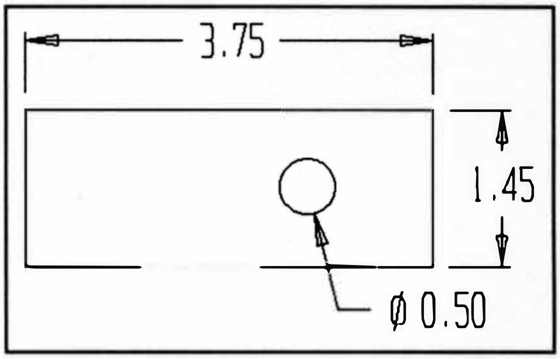
```
do {
      stat = ck_getpos ("Indicate text position for horizontal dimension",1);
   }  while (stat == CK_RETURN);

   if (stat == CK_ESCAPE) {
      return (0);
   }

    if (stat == CK_BACKUP) {
       goto get1;
   }
   opt = stat;

   ck_getcoord (NULL, pos, NULL, NULL);
   refln.x1 = x1;
   refln.y1 = y2;
   refln.x2 = x2;
   refln.y2 = y2;
```

```
    ck_lindim (pos[0], pos[1], 0.0, NULL, &refln, 0.0, CK_LDM_HORZ,
        NULL, NULL, NULL, &att);

    do {
        stat = ck_getpos ("Indicate text position for vertical dimension",1);
    }   while (stat == CK_RETURN);

    if (stat == CK_ESCAPE) {
         return (0);
    }

     if (stat == CK_BACKUP) {
        goto get1;
    }
    opt = stat;

    ck_getcoord (NULL, pos, NULL, NULL);
    refln.x1 = x2;
    refln.y1 = y1;
    refln.x2 = x2;
    refln.y2 = y2;

    ck_lindim (pos[0], pos[1], 0.0, NULL, &refln, 0.0, CK_LDM_VERT, NULL,
    NULL, NULL, &att);

    goto get1;
}
```

## Appendix D: Example of Dimensions for Rectangle and Circle

Description: This program furthered the study of dimensions with CDE modules. After creating the rectangle, this program asks the user for placement of the horizontal and vertical dimensions for the rectangle and then will also dimension a circle that is placed by the user. The text position of the circle dimension is also determined by the user.

---



**Output of program is shown above.**

```
#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"

int
dimension_1 ()
{
    double pos[3];
    int opt, stat, dblevel;
    double x1, y1, z1, x2, y2, z2;
    CK_ENTATT att;
    CK_REFLN refln;
    CK_REFARC refarc;
    double x,y;

    do {
         stat = ck_getpos ("Indicate text position for horizontal dimension",1);
    }   while (stat == CK_RETURN);

    if (stat == CK_ESCAPE) {
          return (0);
    }

     if (stat == CK_BACKUP) {
        goto get1;
    }
    opt = stat;

    ck_getcoord (NULL, pos, NULL, NULL);
    refln.x1 = x1;
    refln.y1 = y2;
    refln.x2 = x2;
    refln.y2 = y2;
```

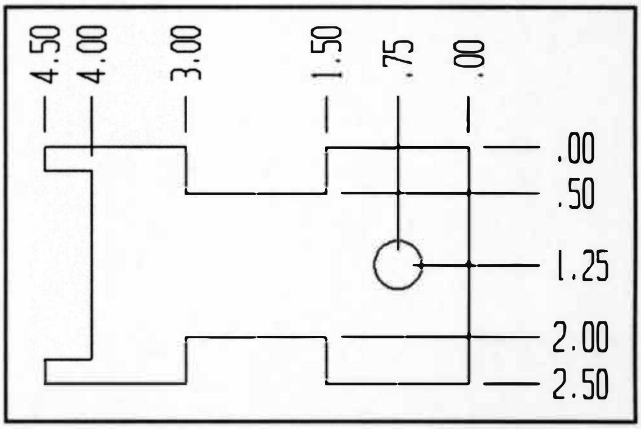
```
ck_lindim (pos[0], pos[1], 0.0, NULL, &refln, 0.0, CK_LDM_HORZ,
    NULL, NULL, NULL, &att);

do {
    stat = ck_getpos ("Indicate text position for vertical dimension",1);
}  while (stat == CK_RETURN);

if (stat == CK_ESCAPE) {
     return (0);
}

 if (stat == CK_BACKUP) {
    goto get1;
}
opt = stat;

ck_getcoord (NULL, pos, NULL, NULL);
refln.x1 = x2;
refln.y1 = y1;
refln.x2 = x2;
refln.y2 = y2;

ck_lindim (pos[0], pos[1], 0.0, NULL, &refln, 0.0, CK_LDM_VERT, NULL,
NULL, NULL, &att);

do {
    stat = ck_getpos ("Indicate position for hole",1);
}  while (stat == CK_RETURN);

if (stat == CK_ESCAPE) {
      return (0);
}

 if (stat == CK_BACKUP) {
    goto get1;
}
opt = stat;
ck_getvar ("@xworld", &x);
ck_getvar ("@yworld", &y);

ck_circle (x, y, 0.0, 0.25, &att);

do {
        stat = ck_getpos ("Indicate text position for hole dimension",1);
    }   while (stat == CK_RETURN);

    if (stat == CK_ESCAPE) {
          return (0);
    }

     if (stat == CK_BACKUP) {
        goto get1;
    }
    opt = stat;

    ck_getcoord (NULL, pos, NULL, NULL);
    refarc.sang = 0.0;
    refarc.dang = 360.0;
    refarc.x = x+.25;
    refarc.y = y;
    refarc.rad = .25;

    ck_cirdim (pos[0], pos [1], 0.0, NULL, &refarc, NULL, CK_CDM_DIAM,
    NULL, NULL, NULL, &att);

    goto get1;
}
```

## Appendix D: Examples of Initial Programs - Ordinate Dimensions

Description: This program was also created to test the dimensioning capabilities of the CDE module but it used ordinate dimensioning instead of standard horizonatl dimensions. It will prompt the user for the base ordinate position and then successive positions to dimension from this base position. The program will keep prompting the user for the next position until the RETURN key is pressed.

---

```
/* hrzordim.c  */

#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"

void
horz_orddim ()
{
 int  stat, num, i, defopt = 1;
 double base_pos[3],
        text_pos[3], pos[3];
 unsigned long idlst[10];
 CK_LOCDEF locdef[10];
 CK_REFPNT refpnt[3];
```



**Output of program is shown above.**
(dimensions only)

```
get_base_pos:
     stat = ck_getpos_def ("Indicate base ordinate position",
     defopt, &locdef[0]);

    if ((stat == CK_ESCAPE) || (stat == CK_BACKUP)) {
        return;
    }

    defopt = stat;
    ck_getcoord (NULL, base_pos, NULL, NULL);

    stat = ck_getpos ("Indicate text position", 1);
    if ((stat == CK_ESCAPE) || (stat == CK_BACKUP)) {
      return;
}
ck_getcoord (NULL, text_pos, NULL, NULL);

num = 0;
refpnt[0].x = base_pos[0];
refpnt[0].y = base_pos[1];

refpnt[1].x = text_pos[0];
refpnt[1].y = text_pos[1];

refpnt[2].x = base_pos[0];
refpnt[2].y = base_pos[1];

ck_orddim (text_pos[0], text_pos[1], 0.0, NULL, refpnt, 0.0,
     CK_ODM_HORZ,NULL, NULL, NULL, NULL);
ck_inquire (CK_INQ_LASTID, &idlst[num++]);

do {
 stat = ck_getpos_def ("Indicate position (RET to end)",
 defopt,&locdef[num]);

 if ((stat == CK_ESCAPE) || (stat == CK_BACKUP)) {
     ck_del_entity (CK_ENT_LIST, num, idlst);
     return;
 }
 if (stat == CK_RETURN) {
     break;
 }
 defopt = stat;
 ck_getcoord (NULL, pos, NULL, NULL);

 refpnt[0].x = base_pos[0];
 refpnt[0].y = base_pos[1];

 refpnt[1].x = text_pos[0];
 refpnt[1].y = text_pos[1];

 refpnt[2].x = pos[0];
 refpnt[2].y = pos[1];

 ck_orddim (text_pos[0], text_pos[1], 0.0, NULL, refpnt, 0.0,
 CK_ODM_HORZ,NULL, NULL, NULL, NULL);
```

```
ck_inquire (CK_INQ_LASTID, &idlst[num++]);
   } while (num < 10);

   ck_makecoll (num, idlst);
   for (i = 0; i < num; i++) {
        stat = ck_set_locdef (idlst[i], 1, &locdef[i]);
   }
   goto get_base_pos;
}
```

## Appendix D: Examples of Initial Programs - Open External File

Description: This program was created to test the external capabilities of the CDE module. It was necessary to extract information from an external data file. This program opened an external file named 'frog.dat'. If the file was not found or had problems opening, a prompt was issued in CADKEY that stated it was not able to open this file. If the fie opens, text is places in the prompt area in CADKEY that says 'The file opened!'.

---

```
#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"

int
open_one ()
{
   FILE *ioptr;
   CK_ENTATT att;

   ck_pause("This is first...Press Return");

   ioptr = fopen("c:\\frog.dat", "r");
   if (ioptr == (FILE *)NULL)
   {
       ck_pause("Sorry, cannot open file...Press RETURN!");

   }
   ck_pause("The file opened!");

}
```

Data File: Frog.dat

```
6.0625, 25.8981, 0.0
13.9375, 25.8981, 0.0
10.0, 22.0, 0.0
6.625, 24.25, 0.0
7.0, 25.125, 0.0
```

## Appendix D: Examples of Initial Programs - Opening File and Use of Variables

Description: This program is an extension of the previous program. After opening the external data file, it was necessary to extract information from this file and use this information for data positions within the CADKEY environment. The information in the external file was read one line at a time. This line came into this program as all one variable. It was necessary to take this variable and extract the three coordinate positions from it. This was done by use of the sscanf C function. Once these three positions were found, a line was created using these positions in the CADKEY eenvironment.

---

```
#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"

int
open_one ()
{
   FILE *ioptr;
   char input_line[25];
   float x, y, z;
   CK_ENTATT attr;
   ck_pause("This is first...Press          Return");

   ioptr = fopen("c:\\frog.dat", "r");
   if (ioptr == (FILE *)NULL)
   {
      ck_pause("Sorry, cannot open file...Press RETURN!");
   }
   while( fgets( input_line, 25, ioptr ) != (FILE *)NULL)
   {
      sscanf (input_line, "%f, %f, %f", &x, &y, &z);
      ck_line (0.0, 0.0, 0.0, x, y, z, &attr);

      ck_pause(input_line);

   }
}
```

## Appendix D: Examples of Initial Programs - Data File and Dimensions

Description: The goal of the final program is to extract X, Y and Z coordinates from an external data file and then use these positions form creating dimensions. This program is an extension of the previous program. This program will extract the first two lines of data from the external file and then use these points to create a horizontal dimension in the CADKEY environment. From this point, other types of dimensions were tested which included ordinate dimensions.

---

```
#include <stdio.h>
#include "c:\gccsdk\include\ck_cdl.h"

int
open_one ()
{
   FILE *ioptr;
   char input_line[25];
   float x, y, z;
   CK_ENTATT attr = { CK_LNG_DFLT, 5, CK_INT_DFLT,
CK_INT_DFLT,
    199,CK_INT_DFLT, CK_INT_DFLT, CK_INT_DFLT, CK_INT_DFLT };
   int counter;
   CK_REFLN refln;
   CK_REFPNT refpnt[2];
   ck_set(CK_SET_COLOR, 5);
   ck_set(CK_SET_DEC_FR, 3);
   ck_set(CK_SET_DIMMODE, 2);
   ck_set(CK_SET_LDZERO, 1);
   ck_set(CK_SET_TRZERO, 1);
   ck_set(CK_SET_LEVEL, 199);

   counter = 1;

   ioptr = fopen("c:\\frog.dat", "r");
   if (ioptr == (FILE *)NULL)
   {
       ck_pause("Sorry, cannot open file...Press RETURN!");
   }
   counter = 1;
file_extract:
    while( fgets( input_line, 250, ioptr ) != (FILE *)NULL)
   {
       sscanf (input_line, "%f, %f", &x, &y);
   if (counter == 1)
   {
       refln.x1 = x;
       refln.y1 = y;
   }
   if (counter == 2)
   {
       refln.x2 = x;
       refln.y2 = y;

    ck_lindim (10.0, 29.0, 0.0, NULL, &refln, 0.0, CK_LDM_HORZ,
        NULL, NULL, NULL, &attr);
   }
    counter++;

   }
}
```

# Appendix E

## D-M-E Mold Layout

# 7⅞ x 7⅞ D-M-E Standard A-Series Mold Bases

**GENERAL DIMENSIONS**

D = DIAMETER OF LOCATING RING
Cat. No. [illegible] (D = [illegible]) Standard
Cat. No. [illegible] (D = [illegible]) Clamp Type
(For other rings, see pages [illegible])

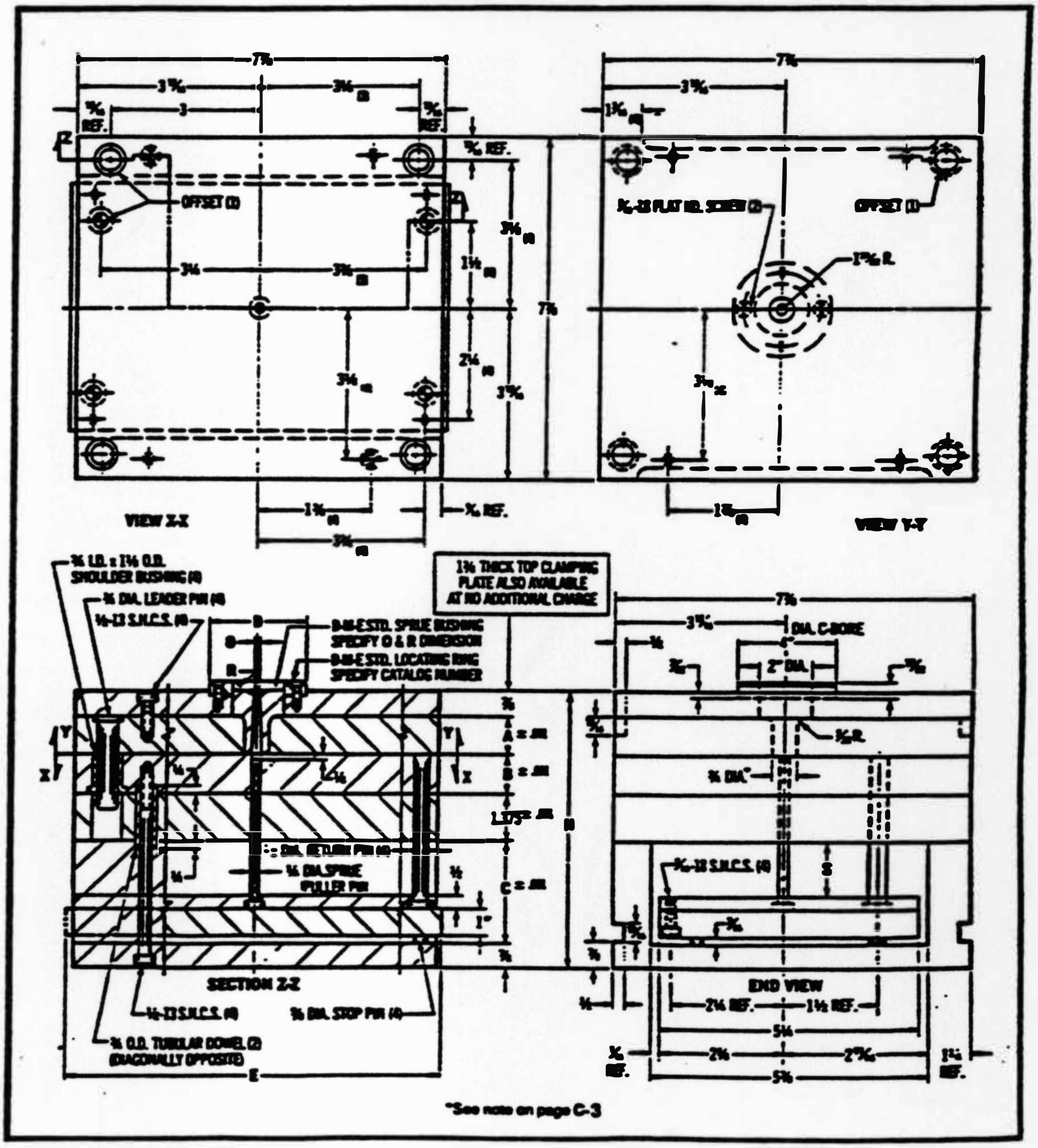
E = LENGTH OF EJECTOR BAR
7⅞, 11⅞, 15" or 23"

O = SMALL DIA. OF SPRUE BUSHING ORIFICE
[illegible]

R = SPHERICAL RADIUS OF SPRUE BUSHING
½ or ¾

| EJECTOR STROKE DATA | | | | | |
|---|---|---|---|---|---|
| C | 2½ | 3" | 3½ | 4"* | 4½"* |
| S | 1⁵⁄₁₆ | 1⁷⁄₁₆ | 1¹⁵⁄₁₆ | 2⁷⁄₁₆ | 2¹⁵⁄₁₆ |

C = Height of Riser
S = Maximum Stroke of Ejector Bar
[illegible]
[illegible]



*See note on page C-3

# Appendix F

## Data File Format

## Appendix E: Format of Data File

Description: This is an example of the data file that is created automatically by the Ultimate® software. This is the file that is read to dimension the mold base. It is divided into four different areas that pertain to the four views of the mold base that are generated by the Ultimate® software.

---

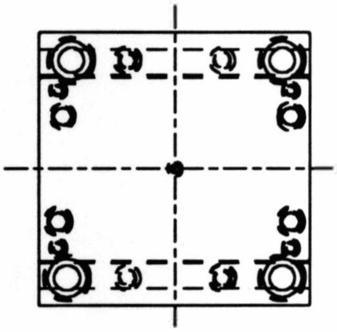
```
BEGIN VIEW "B PLAN-VIEW"
Name = "B PLAN-VIEW"
Type = PLAN
Projection = RIGHT
Datum = (20.000, 44.000)
Width = 12.0
Height = 10.875
BEGIN DIMENSIONING
(SPRUEPULLERPIN,STRAIGHT,"Sprue Puller
Pin",EJECTORCOVERPLATE,"Ejector
Retainer Plate",20.000,44.000)
(RETURNPIN,STRAIGHT,"Return Pin",EJECTORCOVERPLATE,"Ejector
RetainerPlate",25.375,46.813)
(RETURNPIN,STRAIGHT,"Return Pin",EJECTORCOVERPLATE,"Ejector
RetainerPlate",25.375,41.187)
(RETURNPIN,STRAIGHT,"Return Pin",EJECTORCOVERPLATE,"Ejector
RetainerPlate",14.625,41.187)
(RETURNPIN,STRAIGHT,"Return Pin",EJECTORCOVERPLATE,"Ejector
RetainerPlate",14.750,46.813)
(SETSCREW,MOVINGHALF,"Socket Head Cap
Screw",EJECTOR_HOUSING,"EjectorHousing",23.188,48.469)
(SETSCREW,MOVINGHALF,"Socket Head Cap
Screw",EJECTOR_HOUSING,"EjectorHousing",16.812,39.531)
(SETSCREW,MOVINGHALF,"Socket Head Cap
Screw",EJECTOR_HOUSING,"EjectorHousing",23.188,39.531)
(SETSCREW,MOVINGHALF,"Socket Head Cap
Screw",EJECTOR_HOUSING,"EjectorHousing",16.812,48.469)
(STOPBUTTON,NOTYPE,"Stop Button",EJECTOR_HOUSING,"Ejector
Housing",14.625,41.187)
(STOPBUTTON,NOTYPE,"Stop Button",EJECTOR_HOUSING,"Ejector
Housing",20.000,41.187)
(STOPBUTTON,NOTYPE,"Stop Button",EJECTOR_HOUSING,"Ejector
Housing",25.375,41.187)
(STOPBUTTON,NOTYPE,"Stop Button",EJECTOR_HOUSING,"Ejector
Housing",25.375,46.813)
```

```
(STOPBUTTON,NOTYPE,"Stop Button",EJECTOR_HOUSING,"Ejector
Housing",20.000,46.813)
(STOPBUTTON,NOTYPE,"Stop Button",EJECTOR_HOUSING,"Ejector
Housing",14.750,46.813)
(GUIDEPILLAR,STRAIGHT,"Leader Pin",CAVITYPLATE,"A
Plate",25.125,39.437)
(GUIDEPILLAR,STRAIGHT,"Leader Pin",CAVITYPLATE,"A
Plate",25.125,48.563)
(GUIDEPILLAR,STRAIGHT,"Leader Pin",CAVITYPLATE,"A
Plate",14.875,39.437)
(GUIDEPILLAR,STRAIGHT,"Leader Pin",CAVITYPLATE,"A
Plate",15.062,48.563)
(SETSCREW,EJECTORPLATE,"Socket Head Cap
Screw",EJECTORPLATE,"EjectorPlate",24.563,47.313)
(SETSCREW,EJECTORPLATE,"Socket Head Cap
Screw",EJECTORPLATE,"EjectorPlate",24.563,40.687)
(SETSCREW,EJECTORPLATE,"Socket Head Cap
Screw",EJECTORPLATE,"EjectorPlate",15.437,47.313)
(SETSCREW,EJECTORPLATE,"Socket Head Cap
Screw",EJECTORPLATE,"Ejector Plate",15.437,40.687)
END DIMENSIONING
END VIEW "B PLAN-VIEW"
BEGIN VIEW "A PLAN-VIEW"
Name = "A PLAN-VIEW"
Type = PLAN
Projection = LEFT
Datum = (61.4375, 44.000)
Width = 12.0
Height = 10.875
BEGIN DIMENSIONING
(SPRUEBUSH,NOTYPE,"Sprue Bushing",CLAMPINGPLATE,"Top Clamp
Plate",62.000,44.000)
(LOCATIONRING,STANDARD,"Locating Ring",CLAMPINGPLATE,"Top
ClampPlate",62.000,44.000)
(SETSCREW,FIXEDHALF,"Socket Head Cap Screw",CLAMPINGPLATE,"Top
ClampPlate",58.812,48.469)
(SETSCREW,FIXEDHALF,"Socket Head Cap Screw",CLAMPINGPLATE,"Top
ClampPlate",58.812,39.531)
(SETSCREW,FIXEDHALF,"Socket Head Cap Screw",CLAMPINGPLATE,"Top
ClampPlate",65.188,39.531)
(SETSCREW,FIXEDHALF,"Socket Head Cap Screw",CLAMPINGPLATE,"Top
ClampPlate",65.188,48.469)
(GUIDEPILLAR,STRAIGHT,"Leader Pin",CAVITYPLATE,"A
Plate",56.875,39.437)
(GUIDEPILLAR,STRAIGHT,"Leader Pin",CAVITYPLATE,"A
Plate",56.875,48.563)
(GUIDEPILLAR,STRAIGHT,"Leader Pin",CAVITYPLATE,"A
Plate",67.125,39.437)
(GUIDEPILLAR,STRAIGHT,"Leader Pin",CAVITYPLATE,"A
Plate",66.938,48.563)
(CLOSED_CLAMP_SLOT,NOTYPE,"Closed Clamp Slot",CAVITYPLATE,"A
Plate",61.4375,38.875)
(CLOSED_CLAMP_SLOT,NOTYPE,"Closed Clamp Slot",CAVITYPLATE,"A
Plate",61.4375,48.8125)
END DIMENSIONING
END VIEW "A PLAN-VIEW"
BEGIN VIEW "STACK-UP LENGTH"
Name = "STACK-UP LENGTH"
Type = SECTION
Projection = FRONT
Datum = (20.000, 20.000)
Width = 12.0
Height = 7.875
BEGIN DIMENSIONING
(MOULD-WIDTH,NONE,"",NONE,"",14.0,21.75,26.0,21.75)
(MOULD-HEIGHT,NONE,"",NONE,"",26.0,13.875,26.0,21.75)
END DIMENSIONING
END VIEW "STACK-UP LENGTH"
BEGIN VIEW "STACK-UP WIDTH"
Name = "STACK-UP WIDTH"
Type = SECTION
Projection = BOTTOM
Datum = (62.000, 20.000)
Width = 10.875
Height = 7.875
BEGIN DIMENSIONING
(MOULD-WIDTH,NONE,"",NONE,"",66.875,13.875,56.000,13.875)
(MOULD-HEIGHT,NONE,"",NONE,"",66.875,21.750,66.875,13.875)
(CAVITYPLATE,CLOSED_CLAMP_SLOT,"A Plate",CAVITYPLATE,"A
Plate",56.000,20.875,56.000,20.000)
(COREPLATE,CHAMFEREDPLATE,"B Plate",COREPLATE,"B
Plate",56.000,19.125,56.000,20.000)
(SUPPORTPLATE,CHAMFEREDPLATE,"Support
Plate",SUPPORTPLATE,"SupportPlate",56.000,17.250,56.000,19.125)
(EJECTOR_HOUSING,RECTANGULAR_WITH_SL,"Ejector
Housing",EJECTOR_HOUSING,"EjectorHousing",56.000,13.875,56.000,14.750,
56.000,15.563)
(EJECTOR_HOUSING,RECTANGULAR_WITH_SL,"Ejector
Housing",EJECTOR_HOUSING,"EjectorHousing",57.6875,14.75,65.1875,14.75,
57.75,14.9774,65.125,14.9774)
```

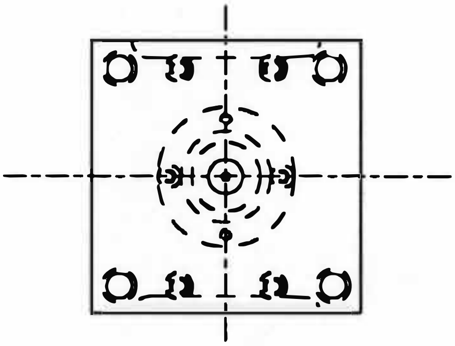
```
(CLAMPINGPLATE,CHAMFEREDPLATE,"Top  Clamp
Plate",CLAMPINGPLATE,"TopClamp Plate",56.000,21.750,56.000,20.875)
(EJECTORPLATE,OVERHUNGPLATE,"Ejector
Plate",EJECTORPLATE,"EjectorPlate",65.0856,15.938,65.0856,14.938)
(EJECTORCOVERPLATE,CHAMFEREDPLATE,"Ejector  Retainer
Plate",EJECTORCOVERPLATE,"Ejector  Retainer
Plate",65.086,16.438,65.0856,15.938)
END DIMENSIONING
END VIEW "STACK-UP WIDTH"
```

# Appendix G

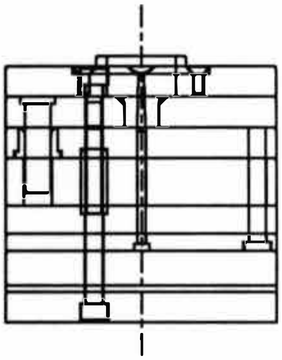
## Ultimate® Layout (Four Views)



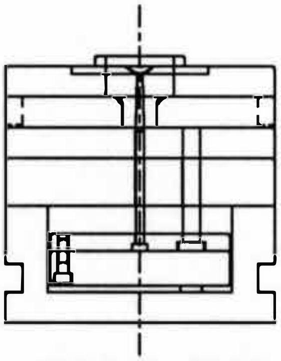
B Plan View



A Plan View



Stack-Up Length



Stack-Up Width

# Appendix H

# Survey Letter

## Appendix H: Survey Letter

February 13, 1995

Dave Williamson
S&K Tool and Die
424 Harrison
Kalamazoo, MI  49007

Dear Mr. Williamson:

I would first like to thank you for speaking to me on the phone and agreeing to participate in a survey which will assist me in completing my Master's thesis project.

As I mentioned on the phone, this project is in conjunction with the new mold design software (Ultimate) offered by The D-M-E Company. This project will enhance the already existing capabilities of the Ultimate software by adding an automatic dimensioning feature to it. I am conducting this survey to accumulate information on industry practices when it comes to creating a mold design layout drawing.

I have included a sample mold layout drawing for your review. As we discussed on the phone, I would like you to look over this mold layout drawing and follow these directions:

1. Review the mold layout drawing produced by the Ultimate software

2. Sketch the necessary dimensions, on this sheet that I have included, to complete the mold layout drawing

**Please include all dimensions as you would for any mold layout drawing that you would produce in your company. Precise values are not necessary. All that is needed are the positions on the mold layout that you would dimension to and from.** (See Figure 1 for an example).

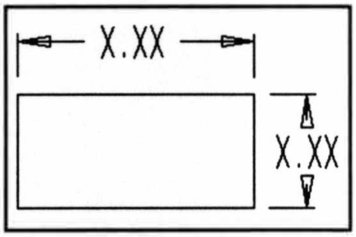


Figure 1: Dimensioning Example

3. Sketch any notes or labels that you would include on a standard mold layout drawing produced by your company

4. If there is anything that has been left out on this drawing, please note this down and send it back to me when you return the layout drawing.

Also, if you have any examples of mold layout drawings that you have done, please send these back with the layout drawing I supplied you with. This would also help me in determining dimensioning practices in industry.

I would appreciate it if you could finish this activity as soon as possible (before February 20th). When you have finished, please use the enclosed envelope to send this layout drawing back to me at Western Michigan University. If you have any questions regarding this drawing or the project in general, please feel free to call me at (616)387-6597.

Again, thank you for your time and input on this project, I appreciate it.

Sincerely,

Cori L. Brown
Graduate Student
Western Michigan University

enc/layout

# Appendix I

## Final Program

## Appendix I: Final Mold Base Program

```
#include < stdio.h >
#include < string.h >
#include "c:\gccsdk\include\ck_cdl.h"

#define TOL 0.001
#define GTR(a, b) ((a)-(b) > TOL)
#define LTR(a, b) GTR(b, a)
#define EQR(a, b) (abs((a) - (b)) < TOL)

typedef struct {
  float x, y, x1, y1, x2, y2, x3, y3;
  } T_dimension_location;
int get_dimension_location(char *line, float *x, float *y);
int get_dimension_location_two(char *line, float *x, float *y, float *x1, float
*y1);
int get_dimension_location_three(char *line, float *x, float *y, float *x1, float
*y1, float *x2, float *y2);
int get_dimension_location_four(char *line, float *x, float *y, float *x1, float *y1,
float *x2, float *y2, float *x3, float *y3);
char *copyString(char *string);
void readDimensions(FILE *ioptr, T_dimension_location *locations, int *number);
void readDimensions_two(FILE *ioptr, T_dimension_location *locations, int
*number);
int compareOnX(T_dimension_location *key, T_dimension_location *elem);
int compareOnY(T_dimension_location *key, T_dimension_location *elem);
int Get_horizontal_locations(T_dimension_location *locations,
T_dimension_location *horiz_locations, int number);
int Get_vertical_locations(T_dimension_location *locations, T_dimension_location
*vert_locations, int number);


/*char *findxy (char *string);*/
char *ptr;
/*int counter;*/
#define ELEMENTS(a) (sizeof(a) / sizeof(a[0]))

/*int main (int argc, char **argv)*/
int auto_dimension ()
        {
        FILE *ioptr = NULL;
        char input_line[250], c = '(', e = '=';
        float xd, yd, width, height, xp, yp, xp1, yp1, xp2, yp2, xp3, yp3, clamp;
        T_dimension_location locations[200], horiz_locations[200],
                        vert_locations[200];
        int max_locations = ELEMENTS(locations), i;
        CK_REFPNT refpnt[3];
        CK_REFLN refln;
        CK_ENTATT attr = { CK_LNG_DFLT, 5, CK_INT_DFLT,
CK_INT_DFLT, 199,
                        CK_INT_DFLT, CK_INT_DFLT, CK_INT_DFLT,
CK_INT_DFLT };

        ck_set(CK_SET_COLOR, 5);
        ck_set(CK_SET_DEC_FR, 3);
        ck_set(CK_SET_DIMMODE, 2);
        ck_set(CK_SET_LDZERO, 1);
        ck_set(CK_SET_TRZERO, 1);
        ck_set(CK_SET_LEVEL, 199);

        ioptr = fopen("\\program\\thesis\\data2.cdo", "rt");
        if (ioptr == (FILE *)NULL)
                {
                printf("Sorry, cannot open file...Press RETURN!");
                return 1;
                }

/* B Plan View */

        while (fgets(input_line, 250, ioptr) != NULL
          && !ferror(ioptr) && !feof(ioptr)
          && strstr(input_line, "Datum") == NULL) {
                /*printf(input_line);*/
                }
        ptr = strchr(input_line, c);
        ptr++;
        sscanf(ptr, "%f,%f", &xd, &yd);

        while (fgets(input_line, 250, ioptr) != NULL
          && !ferror(ioptr) && !feof(ioptr)
          && strstr(input_line, "Width") == NULL) {
                /*printf(input_line);*/
                }
        ptr = strchr(input_line, e);
        ptr++;
        sscanf(ptr, "%f", &width);


        while (fgets(input_line, 250, ioptr) != NULL
          && !ferror(ioptr) && !feof(ioptr)
          && strstr(input_line, "Height") == NULL) {
                /*printf(input_line);*/
                }
        ptr = strchr(input_line, e);
        ptr++;
        sscanf(ptr, "%f", &height);
/*
          Read in all the dimension data

          Find the start of it.
        */

        while (fgets(input_line, 250, ioptr) != NULL
          && !ferror(ioptr) && !feof(ioptr)
          && strstr(input_line, "BEGIN DIMENSIONING") == NULL) {
                }
        /* read in each line */
        readDimensions(ioptr, locations, &max_locations);
        Get_horizontal_locations(locations, horiz_locations, max_locations);
        Get_vertical_locations(locations, vert_locations, max_locations);

/*      creates overall horizontal and vertical dimensions */
        refln.x1 = xd-width/2.0;
        refln.y1 = yd+height/2.0;
        refln.x2 = xd+width/2.0;
        refln.y2 = yd+height/2.0;

        ck_lindim (xd, yd +(height/2.0 + 3.0), 0.0, NULL, &refln, 0.0,
CK_LDM_HORZ, NULL,
                NULL, NULL, &attr);

        refln.x1 = xd+width/2.0;
        refln.y1 = yd+height/2.0;
        refln.x2 = xd+width/2.0;
        refln.y2 = yd-height/2.0;

       ck_lindim (xd + (width/2.0 +3.0), yd, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
                NULL, NULL, &attr);

/*              creates the base ordinate position */
        refpnt[0].x = xd;
        refpnt[0].y = yd;
        refpnt[1].x = xd;
        refpnt[1].y = yd + (height/2.0 + 1.0);

        refpnt[2].x = xd;
        refpnt[2].y = yd;

        ck_orddim (refpnt[1].x, refpnt[1].y, 0.0, NULL, refpnt, 0.0,
CK_ODM_HORZ,
        NULL, NULL, NULL, &attr);


        for (i = 0; i < max_locations; i++)
          {
/* Horz for upper left */
          if (locations[i].x < xd){
               if(locations[i].y > yd){

               refpnt[0].x = xd;
               refpnt[0].y = yd;

               refpnt[1].x = locations[i].x;
               refpnt[1].y = yd + (height/2.0 + 1.0);

               refpnt[2].x = locations[i].x;
               refpnt[2].y = locations[i].y;

               ck_orddim (refpnt[1].x, refpnt[1].y, 0.0, NULL, refpnt, 0.0,
CK_ODM_HORZ,
               NULL, NULL, NULL, &attr);
               }
               }
         /* Horz for upper right corner */
          if (locations[i].x > xd){
               if(locations[i].y > yd){

               refpnt[0].x = xd;
               refpnt[0].y = yd;

               refpnt[1].x = locations[i].x;
               refpnt[1].y = yd + (height/2.0 + 1.0);

               refpnt[2].x = locations[i].x;
               refpnt[2].y = locations[i].y;

               ck_orddim (refpnt[1].x, refpnt[1].y, 0.0, NULL, refpnt, 0.0,
CK_ODM_HORZ,
                NULL, NULL, NULL, &attr);
                }
                }

  /* Vert for upper right corner */
         refpnt[0].x = xd;
         refpnt[0].y = yd;

         refpnt[1].x = xd + (width/2.0 + 1.0);
         refpnt[1].y = yd;

         refpnt[2].x = xd;
         refpnt[2].y = yd;

         ck_orddim (refpnt[1].x, refpnt[1].y, 0.0, NULL, refpnt, 0.0,
CK_ODM_VERT,
         NULL, NULL, NULL, &attr);


         if (locations[i].x > xd){
              if(locations[i].y > yd){

              refpnt[0].x = xd;
              refpnt[0].y = yd;

              refpnt[1].x = xd + (width/2.0) + 1.0;
              refpnt[1].y = locations[i].y;


              refpnt[2].x = locations[i].x;
              refpnt[2].y = locations[i].y;

              ck_orddim (refpnt[1].x, refpnt[1].y, 0.0, NULL, refpnt, 0.0,
CK_ODM_VERT,
              NULL, NULL, NULL, &attr);
              }
              }

  /* Vert for upper left corner */
         refpnt[0].x = xd;
         refpnt[0].y = yd;

         refpnt[1].x = xd - (width/2.0) - 1.0;
         refpnt[1].y = yd;
```

```
          refpnt[2].x = xd;
          refpnt[2].y = yd;

          ck_orddim (refpnt[1].x, refpnt[1].y, 0.0, NULL, refpnt, 0.0,
CK_ODM_VERT,
          NULL, NULL, NULL, &attr);

          if (locations[i].x < xd){
               if(locations[i].y > yd){

               refpnt[0].x = xd;
               refpnt[0].y = yd;

               refpnt[1].x = xd - (width/2.0) - 1.0;
               refpnt[1].y = locations[i].y;


               refpnt[2].x = locations[i].x;
               refpnt[2].y = locations[i].y;

               ck_orddim (refpnt[1].x, refpnt[1].y, 0.0, NULL, refpnt, 0.0,
CK_ODM_VERT,
               NULL, NULL, NULL, &attr);
               }
               }


          }

/* A-Plan View  */

       while (fgets(input_line, 250, ioptr) != NULL
         && !ferror(ioptr) && !feof(ioptr)
         && strstr(input_line, "Datum") == NULL) {
               }
               ptr = strchr(input_line, c);
               ptr++;
               sscanf(ptr, "%f,%f", &xd, &yd);

               refln.x1 = xd-width/2.0;
               refln.y1 = yd+height/2.0;
               refln.x2 = xd+width/2.0;
               refln.y2 = yd+height/2.0;

               ck_lindim (xd, yd +(height/2.0 + 3.0), 0.0, NULL, &refln, 0.0,
CK_LDM_HORZ, NULL,
              NULL, NULL, &attr);

         refln.x1 = xd+width/2.0;
         refln.y1 = yd+height/2.0;
         refln.x2 = xd+width/2.0;
         refln.y2 = yd-height/2.0;

        ck_lindim (xd + (width/2.0 +3.0), yd, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
              NULL, NULL, &attr);

         refln.x1 = xd;
         refln.y1 = yd+height/2.0;
         refln.x2 = xd+width/2.0;
         refln.y2 = yd+height/2.0;

        ck_lindim (xd + (width/4.0), yd + (height/2.0 +1.0), 0.0, NULL,
&refln, 0.0, CK_LDM_HORZ, NULL,
              NULL, NULL, &attr);

         refln.x1 = xd+width/2.0;
         refln.y1 = yd;
         refln.x2 = xd+width/2.0;
         refln.y2 = yd+width/2.0;

        ck_lindim (xd + (width/2.0+1.0), yd + (height/4.0), 0.0, NULL,
&refln, 0.0, CK_LDM_VERT, NULL,
              NULL, NULL, &attr);

     while (fgets(input_line, 250, ioptr) != NULL
      && !ferror(ioptr) && !feof(ioptr)
      && strstr(input_line, "(CLOSED_CLAMP_SLOT") == NULL)
          {}
      get_dimension_location(input_line, &xp, &yp);
      locations[0].x = xp;
      locations[0].y = yp;

     while (fgets(input_line, 250, ioptr) != NULL
      && !ferror(ioptr) && !feof(ioptr)
      && strstr(input_line, "(CLOSED_CLAMP_SLOT") == NULL)
          {}
      get_dimension_location(input_line, &xp, &yp);
      locations[1].x = xp;
      locations[1].y = yp;
```

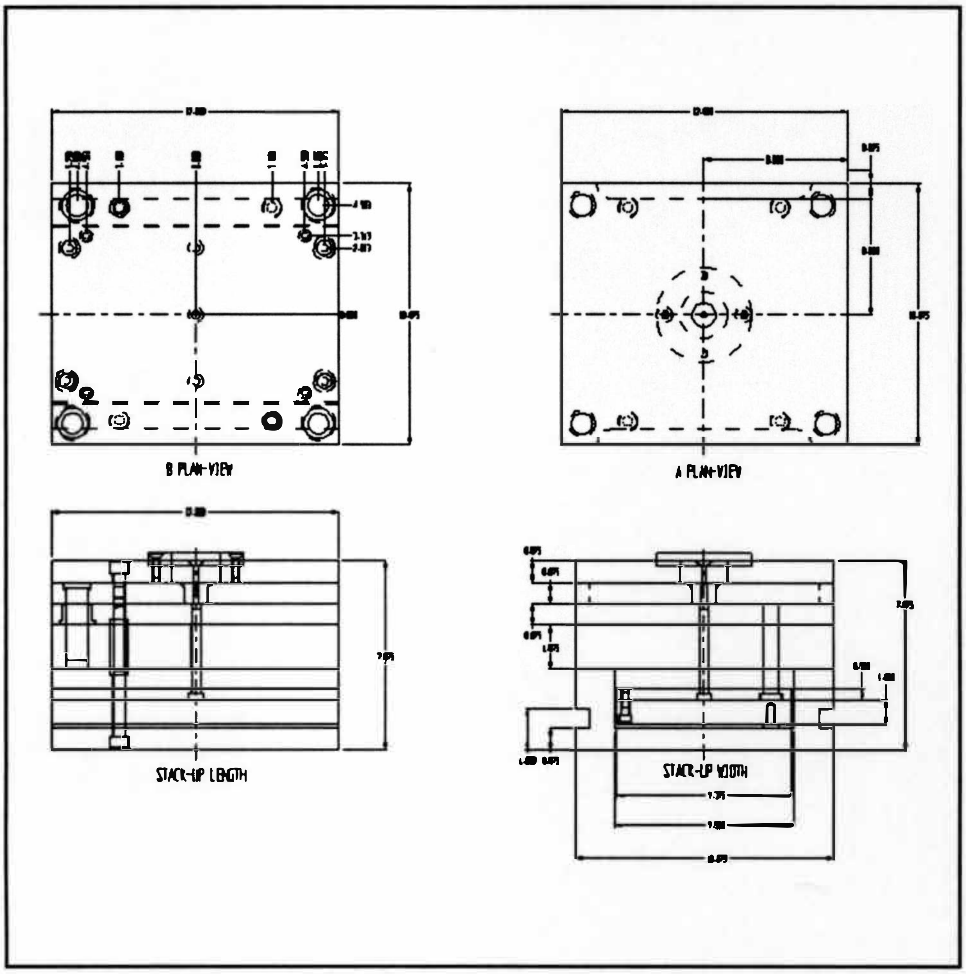
```
          refln.x1 = locations[1].x;
          refln.y1 = locations[1].y;
          refln.x2 = xd+width/2.0;
          refln.y2 = yd+height/2.0;

          ck_lindim (locations[1].x + (width/2.0 + 1.0), yd +(height/2.0 +
          1.5), 0.0, NULL, &refln, 0.0, CK_LDM_VERT,
          NULL, NULL, NULL, &attr);


     if (width == 7.875){
          clamp = 1.1875;
          }
     if ((width == 9.875) && (height == 8.0)) {
          clamp = 1.375;
          }
     if ((width == 9.875) && (height == 11.875)) {
          clamp = 1.3125;
          }
     if ((width == 9.875) && (height == 16.0)) {
          clamp = 1.625;
          }
     if ((width == 9.875) && (height == 20.0)) {
          clamp = 1.375;
          }
     if (width == 10.875) {
          clamp = 1.625;
          }
     if ((width == 10.875) && (height == 12.0)) {
          clamp = 1.5;
          }
     if ((width == 11.875) && (height == 12.0)) {
          clamp = 1.625;
          }
     if ((width == 11.875) && (height == 15.0)) {
          clamp = 1.625;
          }
     if ((width == 11.875) && (height == 20.0)) {
          clamp = 1.875;
          }
     if ((width == 11.875) && (height == 23.5)) {
          clamp = 1.875;
          }
     if(width == 13.375) {
          clamp = 2.1875;
          }
     if ((width == 13.375) && (height == 15.0)) {
          clamp = 1.625;
          }
     if ((width == 14.875) && (height == 17.875)) {
          clamp = 2.3125;
          }
     if ((width == 14.875) && (height == 23.75)) {
          clamp = 2.375;
          }
     if((width == 14.875) &&(height == 29.5)) {
          clamp = 1.4375;
          }
     if (width == 15.875) {
          clamp = 1.625;
          }
     if ((width == 16.5) && (height == 23.75)) {
          clamp = 2.375;
          }
     if ((width == 16.5) && (height == 29.5)) {
          clamp = 1.4375;
          }
     if (width == 17.875) {
          clamp = 1.875;
          }
     if ((width == 19.5) && (height == 23.75)) {
          clamp = 2.0;
          }
     if ((width == 19.5) && (height == 29.5)) {
          clamp = 2.5;
          }
     if (width == 23.75) {
          clamp = 2.25;
          }
     else {
          clamp = 0.0;
          }

     refln.x1 = xd + (width/2.0 - clamp);
     refln.y1 = yd + height/2.0;
     refln.x2 = xd+width/2.0;
     refln.y2 = yd+height/2.0;

/*   ck_lindim (xd + (width/2.0), yd + (height/2.0 + 0.75), 0.0, NULL,
&refln, 0.0, CK_LDM_HORZ,
                NULL, NULL, NULL, &attr);
  */
/*  Stack-Up Length View */

      while (fgets(input_line, 250, ioptr) != NULL
         && !ferror(ioptr) && !feof(ioptr)
         && strstr(input_line, "(MOULD-WIDTH") == NULL)
              {}
         get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
         refln.x1 = xp;
         refln.y1 = yp;
         refln.x2 = xp1;
         refln.y2 = yp1;

         ck_lindim ((xp1 - xp)/2.0 + xp, yp+2.0, 0.0, NULL, &refln, 0.0,
CK_LDM_HORZ, NULL,
                NULL, NULL, &attr);

      while (fgets(input_line, 250, ioptr) != NULL
         && !ferror(ioptr) && !feof(ioptr)
         && strstr(input_line, "(MOULD-HEIGHT") == NULL)
              {}
         get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
         refln.x1 = xp;
         refln.y1 = yp;
         refln.x2 = xp1;
         refln.y2 = yp1;

         ck_lindim (xp1 + 2.0, (yp1 - yp)/2.0 + yp, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
                NULL, NULL, &attr);


/* Stack-Up Width  */

      while (fgets(input_line, 250, ioptr) != NULL
         && !ferror(ioptr) && !feof(ioptr)
         && strstr(input_line, "Datum") == NULL) {

              }
        ptr = strchr(input_line, c);
        ptr++;
        sscanf(ptr, "%f,%f", &xd, &yd);
      while (fgets(input_line, 250, ioptr) != NULL
        && !ferror(ioptr) && !feof(ioptr)
        && strstr(input_line, "Width") == NULL) {

              }
      ptr = strchr(input_line, e);
      ptr++;
      sscanf(ptr, "%f", &width);


      while (fgets(input_line, 250, ioptr) != NULL
        && !ferror(ioptr) && !feof(ioptr)
        && strstr(input_line, "Height") == NULL) {

              }
      ptr = strchr(input_line, e);
      ptr++;
      sscanf(ptr, "%f", &height);

     while (fgets(input_line, 250, ioptr) != NULL
        && !ferror(ioptr) && !feof(ioptr)
        && strstr(input_line, "(MOULD-WIDTH") == NULL)
              {}
        get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
        refln.x1 = xp;
        refln.y1 = yp;
        refln.x2 = xp1;
        refln.y2 = yp1;

        ck_lindim (xd, yp - 4.5, 0.0, NULL, &refln, 0.0, CK_LDM_HORZ,
NULL,
              NULL, NULL, &attr);

     while (fgets(input_line, 250, ioptr) != NULL
        && !ferror(ioptr) && !feof(ioptr)
        && strstr(input_line, "(MOULD-HEIGHT") == NULL)
              {}
        get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
        refln.x1 = xp;
        refln.y1 = yp;
        refln.x2 = xp1;
        refln.y2 = yp1;

        ck_lindim (xp1 + 3.0, yd, 0.0, NULL, &refln, 0.0, CK_LDM_VERT,
      NULL,NULL, NULL, &attr);
```

```
        while (fgets(input_line, 250, ioptr) != NULL
            && !ferror(ioptr) && !feof(ioptr)
            && strstr(input_line, "(CAVITYPLATE") == NULL)
                    {}
            get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
            refln.x1 = xp;
            refln.y1 = yp;
            refln.x2 = xp1;
            refln.y2 = yp1;

            ck_lindim (refln.x1 - 1.0, (yp - yp1)/2.0 + yp, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
                    NULL, NULL, &attr);

          while (fgets(input_line, 250, ioptr) != NULL
            && !ferror(ioptr) && !feof(ioptr)
            && strstr(input_line, "(COREPLATE") == NULL)
                    {}
            get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
            refln.x1 = xp;
            refln.y1 = yp;
            refln.x2 = xp1;
            refln.y2 = yp1;

            ck_lindim (refln.x1 - 1.75, (yp - yp1)/2.0 + yp, 0.0, NULL, &refln,
0.0, CK_LDM_VERT, NULL,
                    NULL, NULL, &attr);

          while (fgets(input_line, 250, ioptr) != NULL
            && !ferror(ioptr) && !feof(ioptr)
            && strstr(input_line, "(SUPPORTPLATE") == NULL)
                    {}
            get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
            refln.x1 = xp;
            refln.y1 = yp;
            refln.x2 = xp1;
            refln.y2 = yp1;

            ck_lindim (refln.x1 - 1.0, (yp1 - yp)/2.0 + yp, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
                    NULL, NULL, &attr);


          while (fgets(input_line, 250, ioptr) != NULL
            && !ferror(ioptr) && !feof(ioptr)
        && strstr(input_line, "(EJECTOR_HOUSING") == NULL)
             {}
        get_dimension_location_three(input_line, &xp, &yp, &xp1, &yp1, &xp2,
             &yp2);
        refln.x1 = xp;
        refln.y1 = yp;
        refln.x2 = xp1;
        refln.y2 = yp1;

        ck_lindim (refln.x1 - 1.0, (yp - yp1)/2.0 + yp, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
             NULL, NULL, &attr);

        refln.x2 = xp2;
        refln.y2 = yp2;

        ck_lindim (refln.x1 - 2.0, (yp - yp1)/2.0 + yp, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
             NULL, NULL, &attr);


      while (fgets(input_line, 250, ioptr) != NULL
        && !ferror(ioptr) && !feof(ioptr)
        && strstr(input_line, "(EJECTOR_HOUSING") == NULL)
             {}
        get_dimension_location_four(input_line, &xp, &yp, &xp1, &yp1, &xp2,
&yp2, &xp3, &yp3);
        refln.x1 = xp;
        refln.y1 = yp;
        refln.x2 = xp1;
        refln.y2 = yp1;

        ck_lindim (xd, yp1 - 4.0, 0.0, NULL, &refln, 0.0, CK_LDM_HORZ,
NULL,
             NULL, NULL, &attr);
        refln.x1 = xp2;
        refln.y1 = yp2;
        refln.x2 = xp3;
        refln.y2 = yp3;

        ck_lindim (xd, yp3 - 3.0, 0.0, NULL, &refln, 0.0, CK_LDM_HORZ,
NULL,
             NULL, NULL, &attr);

      while (fgets(input_line, 250, ioptr) != NULL
      && !ferror(ioptr) && !feof(ioptr)
      && strstr(input_line, "(CLAMPINGPLATE") == NULL)
          {}
      get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
      refln.x1 = xp;
      refln.y1 = yp;
      refln.x2 = xp1;
      refln.y2 = yp1;

      ck_lindim (refln.x1 - 1.75, (yp - yp1)/2.0 + yp, 0.0, NULL, &refln,
0.0, CK_LDM_VERT, NULL,
          NULL, NULL, &attr);

     while (fgets(input_line, 250, ioptr) != NULL
      && !ferror(ioptr) && !feof(ioptr)
      && strstr(input_line, "(EJECTORPLATE") == NULL)
          {}
      get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
      refln.x1 = xp;
      refln.y1 = yp;
      refln.x2 = xp1;
      refln.y2 = yp1;

      ck_lindim (refln.x1 + 4.0, yp + 1.0, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
          NULL, NULL, &attr);


     while (fgets(input_line, 250, ioptr) != NULL
      && !ferror(ioptr) && !feof(ioptr)
      && strstr(input_line, "(EJECTORCOVERPLATE") == NULL)
          {}
      get_dimension_location_two(input_line, &xp, &yp, &xp1, &yp1);
      refln.x1 = xp;
      refln.y1 = yp;
      refln.x2 = xp1;
      refln.y2 = yp1;

      ck_lindim (refln.x1 + 3.0, yp + 1.0, 0.0, NULL, &refln, 0.0,
CK_LDM_VERT, NULL,
          NULL, NULL, &attr);

     fclose(ioptr);
     return (0);
     }
```

```
int get_dimension_location(char *line, float *x, float *y)
{
  char *the_line = NULL;
  char *token = NULL;
  char *first = NULL;
  char *value = NULL;
  int  i;

  if (line == NULL || x == NULL || y == NULL
    || (the_line = copyString(line+1)) == NULL) {
    return 0;
    }
  the_line[strlen(the_line)-2] = '\0';
  /*printf("Look in: %s\n", the_line);*/

  token = strtok(the_line, ",");

/*  printf("Token: %s\n", first);*/

  for (i = 1; i < 5; i++) {
    token = strtok(NULL, ",");
    /*printf("Token: %s\n", token);*/
    }
  value = strtok(NULL, ",");
/*  printf("Read X from %s\n", value); */
  sscanf(value, "%f", x);

  value = strtok(NULL, ",");
/*  printf("Read Y from %s\n", value);*/
  sscanf(value, "%f", y);

  /*printf("X = %f, y = %f\n", *x, *y);*/


  free(the_line);
  return 1;
}

int get_dimension_location_two(char *line, float *x, float *y, float *x1, float *y1)
{
  char *the_line = NULL;
  char *token = NULL;
  char *first = NULL;
  char *value = NULL;
  int  i;
  if (line == NULL || x == NULL || y == NULL
    || (the_line = copyString(line+1)) == NULL) {
    return 0;
    }
  the_line[strlen(the_line)-2] = ’\0’;
  /*printf("Look in: %s\n", the_line);*/

  token = strtok(the_line, ",");

/* printf("Token: %s\n", first);*/

  for (i = 1; i < 5; i++) {
    token = strtok(NULL, ",");
    /*printf("Token: %s\n", token);*/
    }
  value = strtok(NULL, ",");
/* printf("Read X from %s\n", value); */
  sscanf(value, "%f", x);

  value = strtok(NULL, ",");
/* printf("Read Y from %s\n", value);*/
  sscanf(value, "%f", y);

  /*printf("X = %f, y = %f\n", *x, *y);*/
  value = strtok(NULL, ",");
/* printf("Read Y from %s\n", value);*/
  sscanf(value, "%f", x1);

  value = strtok(NULL, ",");
/* printf("Read Y from %s\n", value);*/
  sscanf(value, "%f", y1);

  free(the_line);
  return 1;
}

int get_dimension_location_three(char *line, float *x, float *y, float *x1, float
*y1, float *x2, float *y2)
{
  char *the_line = NULL;
  char *token = NULL;
  char *first = NULL;
  char *value = NULL;
  int i;
  if (line == NULL || x == NULL || y == NULL
    || (the_line = copyString(line+1)) == NULL) {
    return 0;
    }
  the_line[strlen(the_line)-2] = '\0';

  token = strtok(the_line, ",");

  for (i = 1; i < 5; i++) {
    token = strtok(NULL, ",");
    }
  value = strtok(NULL, ",");
  sscanf(value, "%f", x);

  value = strtok(NULL, ",");
  sscanf(value, "%f", y);

  value = strtok(NULL, ",");
  sscanf(value, "%f", x1);

  value = strtok(NULL, ",");
  sscanf(value, "%f", y1);

  value = strtok(NULL, ",");
  sscanf(value, "%f", x2);

  value = strtok(NULL, ",");
  sscanf(value, "%f", y2);

  free(the_line);
  return 1;
}

int get_dimension_location_four(char *line, float *x, float *y, float *x1, float *y1,
         float *x2, float *y2, float *x3, float *y3)
{
  char *the_line = NULL;
  char *token = NULL;
  char *first = NULL;
  char *value = NULL;
  int  i;

  if (line == NULL || x == NULL || y == NULL
    || (the_line = copyString(line+1)) == NULL) {
    return 0;
    }
  the_line[strlen(the_line)-2] = ’\0’;

  token = strtok(the_line, ",");

  for (i = 1; i < 5; i++) {
    token = strtok(NULL, ",");
    }
  value = strtok(NULL, ",");
  sscanf(value, "%f", x);

  value = strtok(NULL, ",");
  sscanf(value, "%f", y);

  value = strtok(NULL, ",");
  sscanf(value, "%f", x1);

  value = strtok(NULL, ",");
  sscanf(value, "%f", y1);

  value = strtok(NULL, ",");
  sscanf(value, "%f", x2);

  value = strtok(NULL, ",");
  sscanf(value, "%f", y2);

  value = strtok(NULL, ",");
  sscanf(value, "%f", x3);

  value = strtok(NULL, ",");
  sscanf(value, "%f", y3);

  free(the_line);
  return 1;
}


char *copyString(char *string)
{
  char *copy = NULL;

  if ((copy = (char *)malloc(strlen(string) + 1)) == NULL) {
    return NULL;
    }
  strcpy(copy, string);
  return copy;
}


int compareFloat(float *key, float *elem)
{
  if (LTR(*key, *elem)) {
    return -1;
    }
  else if (GTR(*key, *elem)) {
    return 1;
    }
  else {
    return 0; /* the same */
    }
}
int compareOnX(T_dimension_location *key, T_dimension_location *elem)
{
  if (LTR(key->x, elem->x)) {
    return -1;
    }
  else if (GTR(key->x, elem->x)) {
    return 1;
    }
  else {
    return 0; /* the same */
    }
}
int compareOnY(T_dimension_location *key, T_dimension_location *elem)
{
  if (LTR(key->y, elem->y)) {
    return -1;
    }
  else if (GTR(key->y, elem->y)) {
    return 1;
    }
  else {
    return 0; /* the same */
    }
}
void readDimensions(FILE *ioptr, T_dimension_location *locations, int *number)
{
  int max_number = *number;
  int i = 0;
  char input_line[250];
  float xp, yp;

        /* read in each line */
        while (fgets(input_line, 250, ioptr) != NULL
          && !ferror(ioptr) && !feof(ioptr)
          && strstr(input_line, "END DIMENSIONING") == NULL
          && get_dimension_location(input_line, &xp, &yp)
          && i < max_number) {
                /*get_dimension_location(ioptr, &xp, &yp);*/
                locations[i].x = xp;
                locations[i].y = yp;
                i++;
                }
        *number = i;
        return;
}

void readDimensions_two(FILE *ioptr, T_dimension_location *locations, int
*number)
{
  int max_number = *number;
  int i = 0;
  char input_line[250];
  float xp, yp;

        /* read in each line */
        while (fgets(input_line, 250, ioptr) != NULL
          && !ferror(ioptr) && !feof(ioptr)
          && strstr(input_line, "END DIMENSIONING") == NULL
          && get_dimension_location(input_line, &xp, &yp)
          && i < max_number) {
                /*get_dimension_location(ioptr, &xp, &yp);*/
                locations[i].x = xp;
                locations[i].y = yp;
                i++;
                }
        *number = i;
        return;
}



int Get_horizontal_locations(T_dimension_location *locations,
T_dimension_location *horiz_locations, int number)
{
  memcpy(horiz_locations, locations, number * sizeof(T_dimension_location));
 return;
}
int Get_vertical_locations(T_dimension_location *locations,
T_dimension_location *vert_locations, int number)
{
  memcpy(vert_locations, locations, number * sizeof(T_dimension_location));
 return;
}
```

# Appendix J

## Final Mold Base Layout


B PLAN-VIEW
A PLAN-VIEW
STACK-UP LENGTH
STACK-UP WIDTH

# BIBLIOGRAPHY

American National Standards Institute. (1983). Dimensioning and Tolerancing, ANSI Y14.5M-1982. New York: The American Society of Mechanical Engineers.

Baum, D. (1992, September 15). Go Totally RAD and Build Apps Faster, Datamation, 38, pp.79-81.

Borland International. (1992). Turbo C++ 3.0 User's Guide, Scotts Valley, CA.

Branch, D., Brown, C., & VanderKooi, M. (1992) Creation of Software for the Mold Base Component Design (Tech. Rep. No. TR-ET485-0492-002). Kalamazoo, MI: Western Michigan University, Department of Engineering Technology.

Brown, Cori. (1994, November). [Interview with Dan Branch, Design Manager of A-Tech Mold].

CADKEY. (1993). Software Development Kit Manual Version 6. Windsor, CT.

CADKEY. (1993). Exploring CADKEY's Open Architecture: Using CADL and CDE's Training Kit. Windsor, CT.

CADKEY. (1992). CADKEY Analysis Manual. Windsor, CT.

Crawford, J.T. (1986). Development and Maintenance of Application Software. Electrical Communication, 60, no.3-4 pp.270-277.

D-M-E Company. (1993). The D-M-E Catalog. Madison Heights, MI.

Earle, J. H. (1991). Graphics for Engineers with CADKEY, Massachusetts: Addison-Wesley Publishing Company.

E D Sales and Service. (December 1994). Mold Software for AutoCAD Release 12. Plastics Engineering, 50, no. 12 pp. 34.

Forbes, G. (1989, May). How Well Integrated? Plastics Technology, 35, pp.71-74.

Hutchison, R. C., & Just, S. B. (1988). Programming Using the C Language, New York: McGraw-Hill.

Jones, C. B., & Shaw, R.C. (1990). Case Studies in Systematic Software Development, New York: Prentice Hall.

Kramer, W., & Kramer, D. (1993). Understanding Autolisp, Programming for Productivity, New York: Delmar Publishers Inc.

Lafore, R. (1991). Object Oriented Programming in Turbo C++, Emeryville, California: Publishers Group West.

Lange, J. C. (1984). Design Dimensioning with Computer Graphics Applications, New York:Marcel Dekker, Inc.

Lehmkuhl, N. K. (1983). FORTRAN 77: A Top-Down Approach. New York: Macmilllan Publishing Co.

Lodge, C. (1988, March). New Software Speeds Part Design, Mold Making. Plastics World, 46 pp. 55-58.

Madsen, D. A., & Shumaker, T. M. (1993). AutoCAD and Its Applications, Release 12. South Holland, Illinois: The Goodheart-Willcox Company, Inc.

Mold Design Software Makes Part Shrinkage Predictable. (1989, January). Modern Plastics, 66, pp.10-11.

Mold Makers Slash Lead Times with CAD (Micro Cadam Plus). (1993, September 24). Machine Design, 65, p. 124.

Montgomery, S. L. (1991). AD/Cycle: IBM's Framework for Application Development and CASE. New York: Van Norstrand Reinhold.

National Tool and Manufacturing Company. (1993). National Electronic Library, A-series Mold Sets and Component Parts, Version 3.01. Kenilworth, NJ.

Software Ventures, Inc. (1992). MF/LINK User's Manual. Kalamazoo, MI.

Taylor, P. (1992, July). Adding Designer Intelligence to CAD Draughting. British Plastics and Rubber, pp. 4-5.

Tecnocad, Ltd. (1992). CAMold - Intelligent Flexibility. Sligo, Ireland.

Vallens, A. (1993, April). Software brings speed, economy to moldmaking. Modern Plastics, pp. 59-61.