EMBRY-RIDDLE
Aeronautical University
SCHOLARLY COMMONS

Theses - Daytona Beach | Dissertations and Theses

8-1996

An Investigation of the Relationships Between the Angle of Mental Rotation Required For Spatial Orientation, Response Times, and Accuracy

Ronald D. Archer
Embry-Riddle Aeronautical University - Daytona Beach

Follow this and additional works at: https://commons.erau.edu/db-theses

 Part of the Aerospace Engineering Commons, and the Aviation Commons

Scholarly Commons Citation

Archer, Ronald D., "An Investigation of the Relationships Between the Angle of Mental Rotation Required For Spatial Orientation, Response Times, and Accuracy" (1996). *Theses - Daytona Beach*. 6.
https://commons.erau.edu/db-theses/6

This thesis is brought to you for free and open access by Embry-Riddle Aeronautical University – Daytona Beach at ERAU Scholarly Commons. It has been accepted for inclusion in the Theses - Daytona Beach collection by an authorized administrator of ERAU Scholarly Commons. For more information, please contact commons@erau.edu.

NOTE TO USERS

Page(s) not included in the original manuscript are unavailable from the author or university. The manuscript was microfilmed as received

83, 85, 104, 127, 154, 157

This reproduction is the best copy available.

UMI®

AN INVESTIGATION OF THE RELATIONSHIPS BETWEEN THE ANGLE OF MENTAL ROTATION REQUIRED FOR SPATIAL ORIENTATION, RESPONSE TIMES, AND ACCURACY.

by

Ronald D. Archer

A Thesis Submitted to the

Aeronautical Science Department

in Partial Fulfillment of the Requirements for the Degree of

Master of Aeronautical Science

Embry-Riddle Aeronautical University

Daytona Beach, Florida

August 1996

UMI Number: EP31936

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.
In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform EP31936
Copyright 2011 by ProQuest LLC
All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Copyright by Ronald Dwayne Archer 1996

All Rights Reserved

AN INVESTIGATION OF THE RELATIONSHIPS
BETWEEN THE ANGLE OF MENTAL ROTATION REQUIRED
FOR SPATIAL ORIENTATION, RESPONSE TIMES, AND ACCURACY.

by

Ronald D. Archer

This thesis was prepared under the direction of the candidate's thesis committee chair, Dr. Gerald Gibb, Department of Aeronautical Science, and has been approved by the members of his thesis committee. It was submitted to the Department of Aeronautical Science and was accepted in partial fulfillment of the requirements for the degree of Master of Aeronautical Science.

THESIS COMMITTEE:

Dr. Gerald Gibb
Chair

Dr. John Wise
Member

Dr. Richard Gibson
Member

MAS Graduate Program Chair

Department Chair, Aeronautical Science

ACKNOWLEDGEMENTS

There are many people behind the successfull completion of this study. First and foremost, I thank the Lord for all of the guidance and blessings that He has provided me. I also want to thank my very supportive family and friends whom have always been behind me throughout my academic endeavors.

I need to express my sincere appreciation to my commitee members, Dr. Gerry Gibb, Dr. Richard Gibson, and Dr. John Wise for their expertise and advice. Additionally, I need to thank to Dr. John Deaton for his assistance in conducting the statistical analysis; Dr. Garland, Mr. Tilden, and Mr. Smith for their assistance in providing participants; and Mr. Banerje for making my experimental design a reality.

ABSTRACT

Author: Ronald D. Archer

Title: An Investigation of the Relationships Between Angle of Mental Rotation Required For Spatial Orientation, Response Times, and Accuracy

Institution: Embry-Riddle Aeronautical University

Degree: Master of Aeronautical Science

Year: 1996

The purpose of this study is to investigate the relationship between the angles of mental rotation when attempting to spatially orientate and the resulting response times and levels of accuracy. By means of a computer program, participants were presented with 64 mental rotational trials. The mental rotational trials consisted of a triangle placed in the center of the screen with a standard stick symbol of an aircraft appearing at various headings and orientations around the triangle. The participants were required to imagine themselves inside the flight deck of the aircraft, and then respond as quickly and accurately as possible to where the triangle is in relation to their orientation. Analysis of the data indicated that as the amount of angular displacement increased from the straight ahead and directly behind positions, the response times and accuracy rates increased and decreased respectively. Additionally, responses for the cardinal orientations were faster than the non-cardinal orientations.

TABLE OF CONTENTS

Appendix

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Anyone who has used standard north-up road maps or navigational charts understands the dilemma of having to either mentally or physically rotate the map/chart in order to help one understand their orientation; where they are, what direction are they going, which way to go or turn, etc. This typically occurs when heading in any direction other than north since the common north-up maps "match" or corresponds to the direction or heading of the person. According to Shepard and Hurwitz (1984), "people generally report that it is easier to interpret a turn as a left or a right turn when the road that leads into that turn has been heading upward on the map (i.e., northward, if the map is itself oriented in the conventional way). Under this condition a turn that goes to the right on the map is a right turn and a turn that goes to the left is a left turn" (p. 172).

Therefore, for the purpose of this study, the mental rotation required to "match" the environment with the person's own orientation is a process that occurs when one attempts to orientate where other objects, places, or people are in relation to themselves. Obviously, this process is important to the aviation industry since spatial orientation is one of the many skills required for pilots and air traffic controllers to effectively and safely perform their navigational duties. It should be irrefutable that the pilot/air traffic controller must have continuos understanding and knowledge of where certain objects or places are in relation to their position, location, and direction.

Statement of the Problem

In the occupations of pilots as well as air traffic controllers, the use of navigational charts and maps are crucial for the user to gain and/or maintain spatial orientation; where they are, where they are heading, and which way to proceed. In the flight decks of many general, corporate, commuter, and commercial aircraft as well as for testing for rental car companies, the implementation of navigational maps have been integrated onto electronic displays. The two general types of electronic navigational maps are north-up and track-up displays. The north-up display is similar to the typical road maps or aeronautical charts in that the direction of north remains at the top of the display, regardless of the heading of the vehicle (aircraft, automobile, boat, etc.). The track-up display is modified so that the map itself rotates in order to correspond with the heading of the vehicle. As concluded by Aretz (1988, 1989), the track-up reduces the amount of mental rotation required since the environment on the map/chart corresponds to the viewpoint of the user. With these concepts in mind, the data from the present study along with the other research will support the use of track-up displays in order to make faster navigational decisions.

Another application of navigation displays becomes prevalent with the many issues being addressed with the redesigning of the air traffic control displays. Currently, the air traffic controller's display is a north-up depiction of a particular sector. The controllers are constantly required to mentally

orientate the position and location for each aircraft and the environment relating to it. In order for the controllers to give directions to each aircraft, they must mentally rotate the environment to match the particular aircraft's so that they can direct which heading or direction for the aircraft to go. With the supporting data from this mental rotation study, another possible application may be for the electronic displays to allow the air traffic controller to rotate the map or display in order to lower the amount of mental rotation required.

There have been several studies investigating the principles of mental rotation, but relatively few have investigated mental rotation in regard to the orientational and navigational considerations mentioned. Therefore, the purpose of this study is to investigate the relationship between the amount of mental rotation required (angle of rotation) and the response times and accuracy required for achieving/maintaining the spatial orientation required for navigational tasks.

Review of Related Literature

Spatial orientation and sense of direction are skills necessary to adequately perform effectively in occupations which require the ability to navigate in an environment such as piloting aircraft, watercraft, and driving automobiles. Kozlowski and Bryant (1977) investigated and defined sense of direction as an "awareness of location or orientation" (p. 590). They found that self-reports of sense of direction were reflective of their spatial orientation ability. Even when orientation was emphasized to the participants, the "good

sense of direction people" showed improved accuracy of their representation of the area, whereas "poor sense of direction people" showed no hint of improved performance. Therefore, Kozlowski and Bryant concluded "that the improved orientation of people with a good sense of direction is not automatic or facile, but it requires possibly both (a) a conscious effort to orient and (b) repeated exposure to an environment" (p. 590).

However, when one is consciously trying to spatially orientate the location of other objects in relation to their position or heading, sometimes mental rotation is required. This mental rotation is an attempt to "match" the actual environment in which one is navigating with the perspective, orientation, or heading of the person. Therefore, when investigating the requirements for spatial orientation during navigational tasks, research of mental rotation becomes necessary.

The majority of mental rotational studies conducted have been based upon the experimental designs of Shepard and Metzler (1971) and Cooper and Shepard (1973). Shepard and Metzler required subjects to make same-different responses to pairs of perspective line drawings depicting unfamiliar, three-dimensional objects. The participants were required to respond with the "same" response when the two objects where the same, regardless of whether they were in the same or different orientations. The "different" responses were required when the pair of objects were mirror-image reversals of each other, again regardless of the same or different orientations. Shepard and Metzler

found that time required for the same-different judgments increased linearly with the angular displacement between the two objects.

The Cooper and Shepard (1973) study, which became the premise for the majority of the mental rotation studies, consisted of the stimulus of a single alphabet or numerical figure (i.e., "R", "G", or "5") which was rotated around in 60 degree increments. In addition to the rotation of the letter, the letter also appeared either mirror-imaged or in standard form. As seen in Figure 1, the participants were asked to respond to whether the rotated letter was of standard or mirror-imaged form, thus requiring the mental rotation of the letter to upright in order to discern the form of the letter.

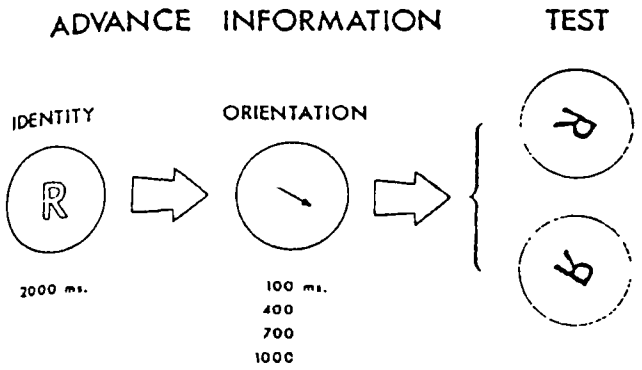


Figure 1. Stimulus for the Cooper & Shepard (1973) Study.

Cooper and Shepard found that the time required for the judgments was nonlinear with the angular displacement of the letter from the 360 degree orientation. More specifically, the results provided evidence that the function relating response time to orientation was symmetrical with respect to the 180

degree orientation (see Figure 2). This function indicated that the stimuli were rotated through the minimum angle necessary to reach upright. Cooper and Shepard also suggested that the nonlinearity may have been due to the concept that mental rotation was not required for stimuli presented at relatively small disorientations from upright. A study conducted by Hock and Tromley (1978) provided a possible explanation by stating that "a familiar stimulus can be perceptually upright even though it is not in its physically upright, or normal, orientation " (p. 529).

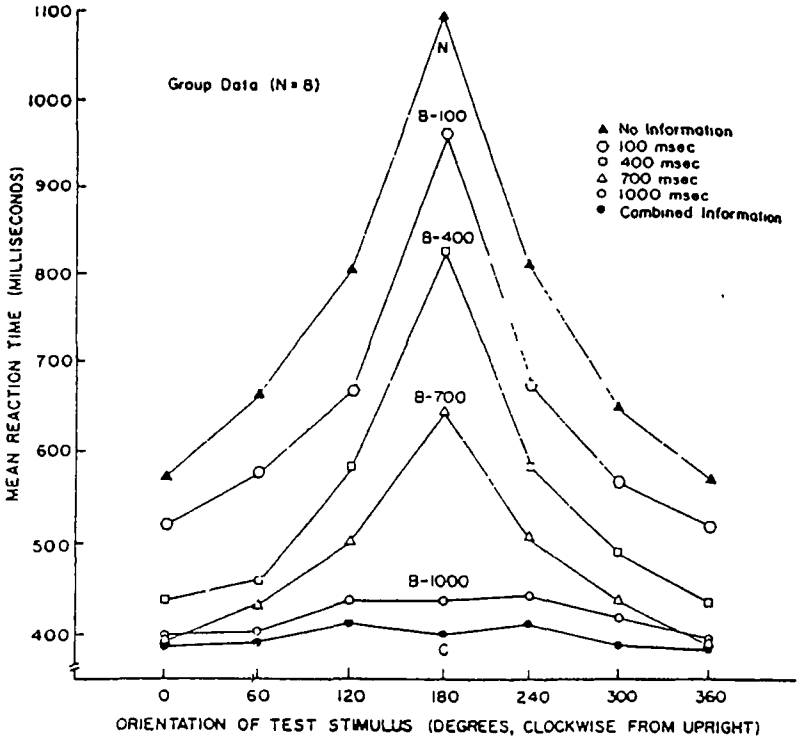


Figure 2. Function of Mean Response Times and Amount of Angular Displacement from Cooper & Shepard (1973).

With this possibility, Hock and Tromley suggested that the observed shape of the letter was an important factor influencing perceptual uprightness. Therefore, they selected letters which were based on their shape. The letters were either circular (e, G), elongated (e, L, J), or rectangular (e, R, F). Even though the letters which were predicted to have a narrow range of perceptual uprightness (e, G) produced a linear rotation function, the letters predicted to have a broad range of perceptual uprightness (e, F, R; L, J) also produced a linear rotation function, but only at orientations outside of their range of perceptual uprightness. Their results supported Cooper and Shepard's suggestion that one of the possible reasons for the nonlinearity of the mental rotation was due to rotation not being required when the orientation of their stimuli were perceptually upright.

However, several other studies (i.e., Hock & Ross, 1975; Cooper & Podgorny, 1976; Maki, 1986; Corballis & Cullen, 1986; and Bethell-Fox & Shepard, 1988) were conducted to investigate the effects of familiarity, similarity, and complexity on mental rotation. The Hock and Ross (1975) study examined the effects of familiarity on mental rotation by requiring the participants to make same-different decisions concerning unfamiliar dot patterns. Based on Hock's (1973) experiment, two dot patterns were simultaneously presented. The "same" responses were required when the two dot patterns were identical, whether they were in same or different orientations. Likewise, the "different" responses were required when the two

dot patterns were not identical, whether in same or different orientations. The familiarity effect was found to be significantly greater when the pairs of patterns were in different orientations. This supported their hypothesis that familiarity would facilitate the mental rotation of the dot patterns.

An example of the studies which investigated the effects of complexity and similarity on mental rotation was one conducted by Bethell-Fox and Shepard (1988). The stimulus used for this experiment consisted of patterns of filled-in squares in a 3x3 matrix. The participants were instructed to inspect the presented matrix until its pattern could be remembered and then to press the "ready" button. The matrix was then immediately replaced by one of the four schematic rotational cues which indicated to the participants whether the remembered pattern was now to be imagined as rotated 90 degrees or 180 degrees , clockwise or counterclockwise. Then, once the participants again pressed the "ready" key, they were to select which one of three presented patterns corresponds to the way the original pattern would be when rotated as specified. The encoding, mental rotation, and comparison of unfamiliar stimuli (patterns of filled-in squares in a 3 x 3 matrix) were found to increase with stimulus complexity (measured by the number of separated pieces constituting each figural pattern). Therefore, the majority of these studies provided support for the premise that the time to mentally rotate a stimulus was dependent on the familiarity and complexity of the stimulus.

Other studies of mental rotation concentrated on the effects of practice. The majority of these studies (e.g., Damos; 1991, chap. 7; Thorndyke & Hayes-Roth, 1982; Jolicoeur, 1985; and Pylyshyn, 1979) comparably resulted with a significant increase in performance, decreases in response times and increases in accuracy rates. However, as remarked by Pylyshyn (1979), "The influence of practice on rotation rate is found routinely in studies such as these, although it has not generally been reported in the literature, since published results are invariably obtained from highly practiced subjects using overlearned stimuli" (p. 26).

Another major area of mental rotational studies pertains to the investigations of hemispheric, clockwise or counterclockwise, differences in the process of conducting mental rotational tasks. As stated by Burton et al. (1992), "The nature of hemispheric specialization for mental rotation in unclear, with some studies indicating a right hemisphere (RH) advantage and others a left hemisphere (LH) advantage" (p. 192). A possible explanation given by this study may be that research has suggested that the previously discussed areas or factors of mental rotation (familiarity, complexity, practice) interacts with the hemispheric process. However, the Burton et al. study did result in interactions which suggested that "clockwise rotations were more readily performed in the left visual field and counterclockwise rotations in the right visual field" (p. 192).

Another study by Cook et al. (1994) suggested that a cooperation takes place between the two hemispheres which perform different functions. They explain that their results support other research findings which found that one hemisphere (usually the LH) actively manipulates its visual information, while the other hemisphere is employed in a reference role. They further stated that both roles are essential for the accurate performance of mental rotation.

Ueker and Obrzut (1993) conducted a study which not only investigated the hemispheric differences, but investigated the possible gender differences for conducting mental rotation. Their mental rotation involved the rotation of a stick figure stimulus which is holding a ball in either the right hand or left hand. The stick figure was then rotated in any of the eight 45 degree orientations and the participants were to respond to which side the figure is holding the ball. However, the results from their study indicated that "there were neither hemispheric nor gender effects found with a mental rotation task" (p. 48). Jones and Anuza (1982) also conducted a study which was not able to find a gender difference.

Based on the Shepard and Metzler (1971) experimental method, the Jones and Anuza study focused on the effects of gender and handedness on mental rotation. They did find that "right-handers tended to respond more rapidly than left-handers" (p. 506). However, in addition to the inability to find a gender difference in the response times as already stated, no sex or handedness differences in error rates or accuracy were found.

Unlike the majority of the studies which investigated gender differences, a study conducted by Berg, Hertzog, and Hunt (1982) found age differences in the speed of conducting mental rotation tasks. Four different age groups participated in a mental rotations task for four consecutive days. They found "significant age differences in the linear function relating median reaction times to degrees of rotation: older subjects had higher intercepts and higher slopes" (p. 95). Additionally, they found no indication that age differences in mental rotation performance would disappear after practice.

With all of the possible aspects studied about mental rotation such as the effects of perceptual uprightness, complexity, and familiarity of the stimuli, effects of practice, hemispheric differences in the process of conducting mental rotation, and the possible differences (i.e., age, gender, etc.) in the speed of conducting mental rotation, it can be easily concluded that there are hardly, if not any, limitations to the study of mental rotation. Additionally, this particular research study investigates the degree of mental rotation which becomes required for spatial orientation. Even though the research previously discussed provides the foundations for the study of mental rotation, the rotation of a letter and the determination of whether or not is mirror-imaged or normal provides little support to the investigation of mental rotation required for spatial orientation. However, the majority of the studies did provide a premise which was defined by Koriat and Norman (1984) as "image rotation" (p. 421). This term designates a strategy in which the image of the

stimulus is first rotated to the upright position in order to make some sort of determination concerning the stimuli, such as its spatial orientation. Koriat and Norman in their 1988 study further suggested that "spatial transformation is normally achieved through image rotation" (p. 93). Therefore, with the principles provided by the studies previously discussed, the investigation of the mental rotation required when making spatial orientational judgments could now be conducted.

A study conducted by Loftus in 1978, concluded with a two step model for comprehending compass directions. For the experiment, the subjects were visually presented with a numeric compass direction between 0 and 350 degrees. The subjects' tasks were to indicate their comprehension of the direction by indicating the representation of it on a blank (not labeled or numbered) compass rose and then to push a key when done. The response times between the presentation of the stimulus and the keypress was then used as an indication of the time required to comprehend the direction. The premise made for this study was that the "functions relating RT to 1) the specific direction presented and 2) the way in which the directional information was orientated can then be used to make inferences about the manner in which compass directions are represented and processed" (p. 416). The results suggested that a direction is understood by a two step process of mental operations.

First, the nearest cardinal heading to the target direction (i.e., north, south, east, or west) is computed, and one mentally rotates in order to "face" in the same cardinal direction. This supports the idea that people tend to orientate cardinal headings faster than non-cardinal headings since the cardinal headings were found to be processed first as a means of orientating the other specified target direction or heading. This will provide the basis for the third hypothesis tested in this study.

Second, the differences between the cardinal direction and the desired target direction is computed and a mental rotation, either clockwise or counterclockwise, is conducted until the desired target direction is orientated and designated. Therefore, even though the Loftus study concluded with a technically different two step process, mental rotation was still found to be present and was required when attempting to orientate the location of the specified target.

Other studies which investigated the mental rotation required for spatial orientation were conducted by Aretz (1988, 1989). Similar to this study, the major goal for the two Aretz studies were to investigate the role of mental rotation in the cognitive processing required during aircraft navigation. A comparison was conducted between the mental alignment of two frames of reference: the ego centered reference frame and the world centered reference frame. These frames of references corresponds respectively to the track-up and north-up types of electronic map displays which were explained

previously in this report. Aretz concluded that the amount of required mental rotation was lower when in the ego centered reference frame, thus producing faster response times in making navigational decisions. Aretz (1989) also found that "mental rotation was most prevalent in the simultaneous trials and diminished considerably in the sequential trials" (p. 11). This supports a finding from a study by Hintzman, O'Dell, and Arndt (1981) which theorized that mental rotation is only required when a visual map, and not when a "cognitive map", (i.e., memory) is used. Therefore, since an electronic map is visually available, mental rotation will be performed when the ego centered reference frame and the world centered reference frame are not aligned.

Hintzman, O'Dell, and Arndt (1981) conducted a series of experiments where the subjects were required to determine the location of targets while trying to imagine themselves facing in various orientations. The study also investigated these orientational tasks when the map is either committed to memory ("cognitive maps") or when it is visually available as stated in the previous paragraph. However, only the visually presented map investigations will be discussed since the possible implications for this study pertain to the use of physical navigational maps, charts, and displays.

Figure 3 shows the stimulus display and response board used for the experiments. The participants were required to imagine themselves facing in the particular direction the arrow and to respond where, in relation to their orientation, the large dot is located. Using the response board (right side of

Figure 3), the participants were to "point to" the orientation corresponding to the location of the large dot. In this example, the large dot is located behind and to the left of the direction of the arrow. Each trial would display a different orientation (the eight 45 degree points around the compass rose) as indicated by the arrow as well as a different target location as indicated by a large dot.

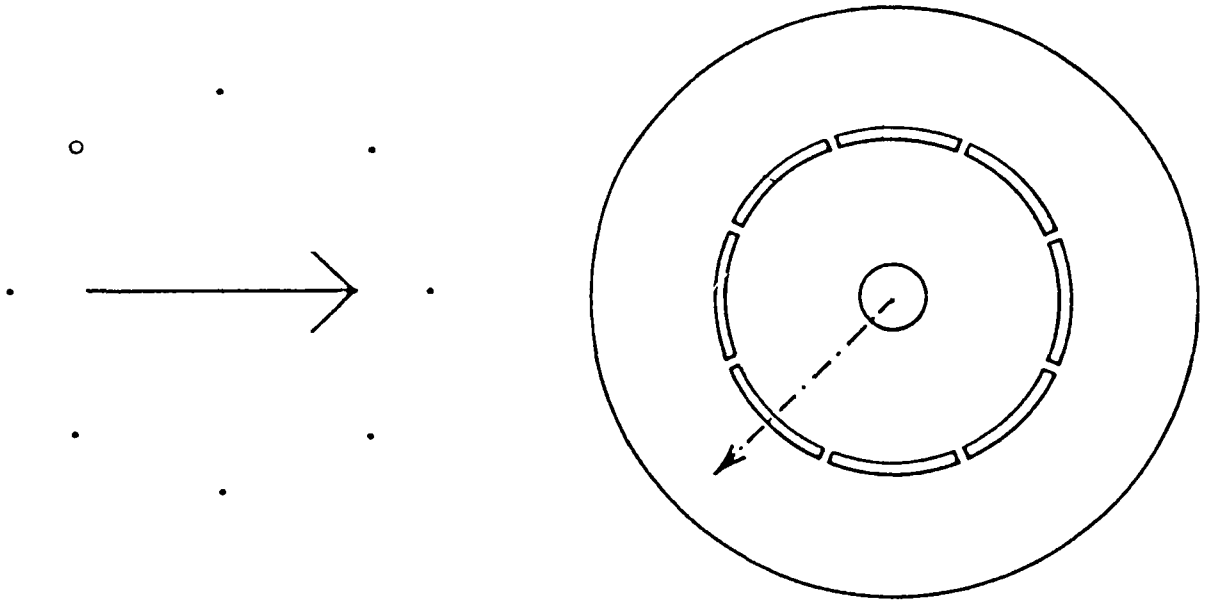


Figure 3. Stimulus for the Hintzman et al. (1981) Study.

The mean response times acquired for the eight 45 degree orientations resulted with a function as shown in Figure 4. As can be seen, the participants responded the fastest when making Front or Back decisions. This supports the premise that the participants orientated quicker at the 360 and 180 orientations since the amount of mental rotation was at its lowest requirement. Therefore, the response times required for the participants to spatially

orientate the location of the target then increased as the amount of required mental rotation was increased from the straight ahead and the directly behind positions. However, as can also be seen by Figure 4, the response times were slightly lower for the "Right" (090) and "Left" (270) orientations as compared to the positions immediately surrounding them. A possible explanation for this occurrence may be that the participants orientated the cardinal directions faster than the non-cardinal directions which is congruent with other research, (i.e., Loftus, 1978).

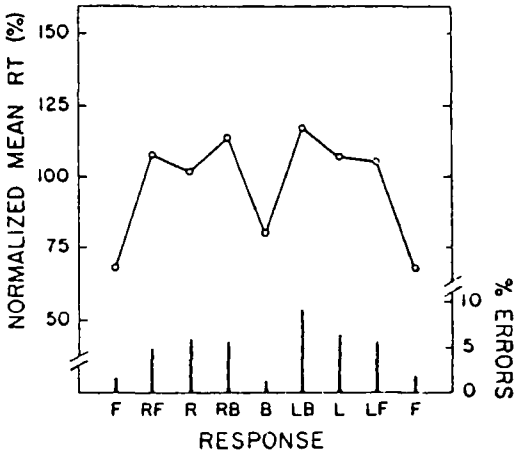


Figure 4. Function of Mean Response Times, Percentage of Errors, and Amount of Angular Displacement from Hintzman et al. (1981).

Figure 4 additionally displays the recorded accuracy rates for each of the eight orientations. An inverse function of the response times, the participants answered the Front (360 degree) and Back (180) most accurately. Therefore, the accuracy rates then decreased as the amount of

required mental rotation was increased from the straight ahead and the directly behind positions. As for the explanation for the accuracy rates being slightly higher for the 090 and 270 positions, it may also be hypothesized that the participants were more accurate when conducting the mental rotations at the cardinal positions than at the non-cardinal positions.

The literature on the topic of mental rotation is extensive. This may be due to the almost unlimited number of parameters associated with mental rotation. As discussed, some of these include familiarity, perceptual uprightness, and complexity of the stimulus, effects of practice, hemispheric differences in the process of conducting mental rotation, and the possible differences (i.e., age, gender, etc.) of mental rotation. However, for the purpose of this study, the number of investigations into the mental rotation required when attempting to spatially orientate are relatively few. Such studies have suggested that an understanding into this realm of mental rotation may help to provide guidelines for designing displays to be used by people performing navigational tasks.

Statement of the Hypothesis

The previous research has shown that larger angles of mental rotation require longer times to process the information in order to orientate. Therefore, it is hypothesized that as the amount of mental rotation required is increased from the straight ahead position (360) and from the directly behind position (180), the response times will similarly increase. Additionally, it is

hypothesized that as the amount of mental rotation required is increased from the straight ahead and from directly behind positions, the accuracy will decrease. The third hypothesis states that the response times will be significantly less for the mental rotation of the cardinal directions (360, 090, 180, and, 270) than for the non-cardinal directions (045, 135, 225, and 315). The fourth hypothesis states that the accuracy rates will be significantly better for the mental rotation of the cardinal directions (360, 090, 180, and, 270) than for the non-cardinal directions (045, 135, 225, and 315).

Method

Subjects

The subjects were 100 students who volunteered from several upper-class level air traffic control (ATC) and flight courses at Embry-Riddle Aeronautical University. The subjects received extra course credit for participating in the experiment. Since most of these students will be employed as pilots or air traffic controllers upon graduation, they can be considered as a subsample of the larger pilot and ATC populations.

Convenience and judgment sampling were possible sources of sampling bias. The limited resources available for sampling produced the major concern for convenience sampling. Also due to the possible limited number of volunteers available at the selected cluster, the question of their representation of the entire population was of concern for judgment sampling bias. Additionally, another bias may be due to the subjects not having as much experience as those in the target population. However, these effects should be small and the results should be considered applicable to the target population.

Instrument

A computer program (Appendix B) was designed to present the stimulus and to record the response times and accuracy of the subjects. The two 486 computers used were located in the same room with a room divider between them to eliminate the possibility of distraction between subjects

participating simultaneously. The second computer was used only when two or more participants arrived for the experiment at the same time. When such an occasion arose, the two participants were simultaneously tested.

The stimulus consisted of a triangle centered in the middle of the screen with a standard stick aircraft symbol randomly appearing at one of the eight 45 degree compass positions around the triangle. The participants were then required to respond by pressing one of the eight corresponding outside keys of the numeric keypad located on the right side of a standard computer keyboard.

Design

The design of the experiment was based from the Hintzman et al. (1981) study. As discussed earlier, the tasks of their participants were to indicate the direction from themselves that the target dot would be if they were in the orientation indicated by the arrow. Likewise, the participants in this study were required to indicate the direction from themselves that the triangle would be if they were in the orientation of the aircraft symbol. The participants responded to the stimulus by pressing the corresponding answer with one of the eight keys on the numeric keypad. All of the other keys on the keyboard were locked out in case the participants were to accidentally strike the wrong key.

The independent variable for the experiment was the amount of mental rotation required for the participants to spatially orientate where the triangle is

located in relation to the heading of the aircraft symbol. The independent variables were categorized by the eight 45 degree points on the standard 360 degree compass rose (360, 045, 090, 135, 180, 225, 270, 315). The order of the presentation of the trials were randomly selected and arranged in a fixed order for all subjects. Each participant completed a total of 64 trials, eight trials for each of the eight variables. The eight trials for each variable were not identical even though the correct responses were the same. The location and direction of the aircraft symbol appeared at all of the eight different headings possible at each of the eight 45 degree positions around the triangle. The dependent variables for the experiment were the response times and accuracy rates recorded.

Procedures

The participants for this study were volunteers from upper-class level courses at Embry-Riddle Aeronautical University. They received extra course credit for participating in the study. The confidentiality of the participants was maintained by identifying the subjects with identification numbers which they selected. Throughout the experiment, the participants were only identifiable through the use of the identification numbers; names were not used in the collection, the analysis, nor the reporting of the results.

After entering their identification numbers, the participants were required to go through a programmed set of instructions, a sample trial, and two practice problems (specified in Appendix B). Once these steps were

completed, the participants completed the 64 random mental rotational/orientational trials. The program was designed so that once a response was given by the participants, the next trial was immediately begun. After half of the trials was completed (32), the program would stop and provide the subjects a break. Once the participants were ready to proceed with the other half of the trials, they were given a ten second countdown to allow them to be prepared when the next trial was given.

Pilot Study

The pilot study consisted of two groups of five participants. The first group of five were allowed to proceed from the beginning to the end of the program without any aid. After the participants completed the experiment, they were allowed to ask any questions and to make any suggestions which would make the experiment more clear and understandable. A couple of problems were discovered from the discussions with the first group of participants. The most common misunderstandings regarding the objective of the test were: 1) the participants thought that they were to respond to how many degrees were needed for them to turn in order to head directly towards the triangle; and 2) the participants thought that they were to simply respond where the aircraft symbol was in relation to the triangle. Additionally, it was suggested that a legend showing the correct corresponding keys in relation to the orientations should be provided (see Appendix A). The mean accuracy for the first group was 48.12%.

Therefore, the second group of five participants received the following changes in addition to the computer program. The legend was taped to the desk to the right of the keyboard for use during the experiment. Additionally, a script of further explanation and directions was written and read to each subject after they completed the set of instructions on the computer program. The screen displayed the sample trial so that the participants could better visualize the objective of the experiment while the script was being read. The script read as follows:

"The first pilot study concluded that a couple of misunderstandings were occurring regarding the objective of the experiment. First, you are not to indicate where the aircraft symbol is in location to the triangle, but you are to respond to where the triangle is located in relation to the orientation of the aircraft. The second misunderstanding was for the participants to indicate how many degrees were needed to turn in order to head towards the triangle. Again, this is incorrect. Please make sure you are responding to where the triangle is located in relation to the heading of the aircraft. This is usually achieved by pretending that you are sitting in the flight deck of the aircraft and heading in the direction of the aircraft."

The participants were then allowed to proceed to the two practice trials on the computer program. If they answered incorrectly on the trials, the computer program indicated what the correct response should have been. If

the participants incorrectly answered both trials, then they were stopped and were read the following script:

"Again, the objective of the experiment is to respond to where the triangle is located in relation to the heading of the aircraft. By looking at the last practice trial, it may help to pretend you are sitting in the flight deck and then identifying where the triangle is located in relation to the nose of the aircraft. In this case, the triangle is behind you and to the left which is at the 225 positions or the #1 key."

The participants were then allowed to proceed with the remaining instructions and actual completion of the 64 trials. With these changes given, the mean accuracy for the second group of five participants was 92.80%. With this significant increase in accuracy between the first and second section of the pilot study, $t(8) = -4.40$, $p < .003$, the study was initiated with the remainder of the volunteers (91) implementing the same procedures used during the second part of the pilot study.

Upon completion of the experiment, the participants who requested to see their results were given the opportunity. They were instructed to return to the location where the experiment was held and, by use of their identification number, they were able to see their response times and accuracy rates.

Analysis

Response Times.

By means of the Statistica statistical analysis computer program, a two-way ANOVA was conducted and found a significant difference between the eight 45 degree orientations, $F(7, 720) = 13.48$, $p < .001$. Table 1 shows the resulting mean response times in seconds for each of the eight 45 degree orientations.

Table 1

Resulting Mean Response Times in Seconds

The eight 45 degree orientations	Mean response times in seconds
360 degree orientation	2.077
045 degree orientation	3.26
090 degree orientation	3.10
135 degree orientation	3.63
180 degree orientation	2.21
225 degree orientation	3.62
270 degree orientation	2.83
315 degree orientation	3.44

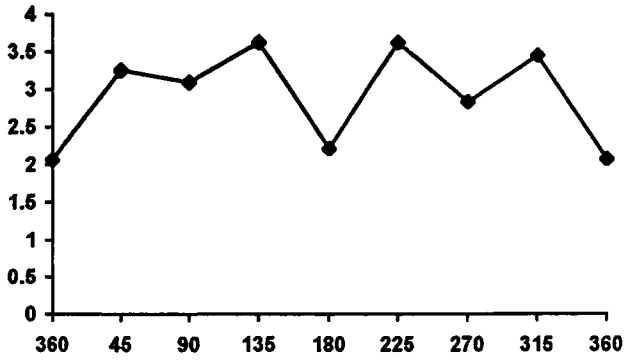


Figure 5. Function of Mean Response Times and Amount of Angular Displacement.

In order to test the hypothesis, there were eight planned comparisons conducted to investigate the relationships between the eight 45 degree positions and the corresponding response times. The order of the comparisons were conducted as the orientations occur clockwise around the compass rose.

The first planned comparison was between the response times of the orientations of the 360 degree position, when the triangle was directed ahead of the aircraft so that no mental rotation was required, and the 045 degree position. The hypothesis was confirmed between these two variables since there was a significant increase in the response times required for the participants to orientate between the 360 position and the 045 position, $F(1, 720) = 25.90$, $p. < .001$.

The second planned comparison of response times was conducted between the orientations of the 045 position and the 090 position. There was not a significant difference between these two positions, $F(1, 720) = 0.48$, $p. < .489$. Even though this comparison is not significantly different, it was found that the participants took longer to orientate at the 045 degree position than at the 090 position. This does not supports the main hypothesis in that the participants did not take longer to mentally rotate and orientate the 090 as compared to the lesser amount of rotation required, the 045. However, this result was anticipated by the third hypothesis, discussed later, which investigated the time required for the participants to mentally rotate and orientate the cardinal versus the non-cardinal headings.

The third planned comparison between the 090 orientation and the 135 degree orientation concluded that there was a significant difference in response times, $F(1, 720) = 5.24$, $p. < .022$. This comparison supports the main hypothesis in that the participants took longer to mentally rotate and orientate at the 135 degree position than at the 090 degree position. This was again expected since the angle of rotation required was higher.

The planned comparison between the 135 orientation and the 180 orientation concluded that there was a significant difference in response times, $F(1, 720) = 37.15$, $p. < .001$. When the triangle was directly behind the aircraft symbol, at the 180 position, the participants were significantly faster at orientating the location of the triangle. This supports the main hypothesis

since it was anticipated that the response would begin to lower as the angle of rotation approached the 180 position. As discussed and supported from the literature, it was common for individuals to mentally rotate up to the 180 degree position. So even though it is numerically higher moving from the 090 to the 180 position, the actual amount of mental rotation becomes less when orientating with items from directly behind. Then, as the angle of rotation proceeds past the 180 position, the actual amount of required mental rotation begins to increase up to the 270 position. From that point, it begins to lower again when approaching the 360 position, or at the straight ahead position. This is further supported by the almost symmetrical formation of mean response times found on Figure 5.

The planned comparison of the response times between the 180 orientation and the 225 orientation also supports the hypothesis since there was a significant increase in the response times, $F(1, 720) = 36.57$, $p. < .001$.

The planned comparison conducted between the 225 orientation and the 270 orientation concluded that there was a significant decrease in response times, $F(1, 720) = 11.71$, $p. < .001$. Likewise with the 090 position, this does not support the main hypothesis since the amount of rotation is increased while the response times decreased. However, in accordance to the third hypothesis, this was also anticipated so that the cardinal headings/positions would require lower response times to mentally rotate and to orientate than with the non-cardinal headings/positions.

The seventh planned comparison was conducted between the 270 orientation and the 315 orientation. The main hypothesis that the response times required to mentally rotate and orientate would increase as the amount of rotation increased was again supported by the significant increase in response times $F(1, 720) = 6.96$, $p. < .008$.

The planned comparison conducted between the 315 orientation and the 360 orientation found a significant decrease in response times, $F(1, 720) = 34.31$, $p. < .001$. Similar to the 180 position, even though the numerical angle of rotation is higher, the actual amount of mental rotation is lower; thus lower response times. This again supported the main hypothesis and the explanation of the symmetrical "M" shaped formation of the mean response times correlating to the angles of rotation (see figure 5).

The final planned comparison for the response times was conducted between the cardinal headings/orientations (360, 090, 180, & 270) and the non-cardinal headings (045, 135, 225, & 315). The third hypothesis which stated that the response times would be lower for the cardinal headings than for the non-cardinal headings was confirmed with a significant difference, $F(1, 720) = 64.53$, $p. < .001$. The mean response times in seconds for the cardinal orientations was 2.56 where as the non-cardinal orientations resulted with a mean of 3.49.

Accuracy. A two-way ANOVA was conducted and a significant difference was found between the eight 45 degree orientations, $F(7, 720) =$

5.32, p.< .001. The resulting mean accuracy rates for each of the eight orientations are shown in Table 2.

Table 2

Resulting Mean Accuracy Rates

The eight 45 degree orientations	Resulting accuracy rates
360 degree orientation	93.96%
045 degree orientation	93.82%
090 degree orientation	86.68%
135 degree orientation	89.69%
180 degree orientation	95.47%
225 degree orientation	93.68%
270 degree orientation	85.44%
315 degree orientation	91.07%

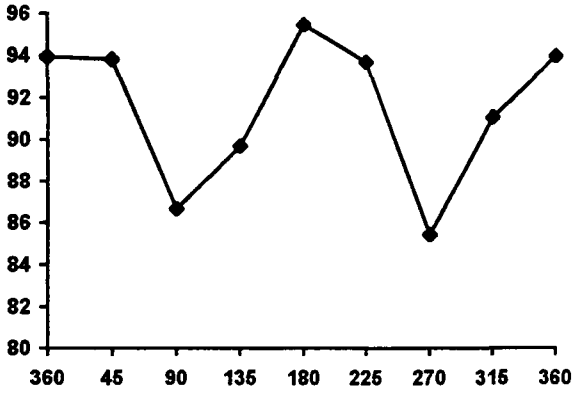


Figure 6. Function of Mean Accuracy Rates and Amount of Angular Displacement.

In order to test the hypothesis, there were eight planned comparisons conducted in order to investigate the relationships between the eight 45 degree positions and the corresponding accuracy rates. The order of the comparisons were conducted as the orientations occur clockwise around the compass rose.

The first planned comparison was between the accuracy rates of the orientations of the 360 degree position, when the triangle was directed ahead of the aircraft so that no mental rotation was required, and the 045 degree position. Even though there was a decrease in the accuracy rates, the hypothesis was not confirmed between these two variables since there was not a significant decrease in the accuracy rates when the participants mentally rotated between the 360 orientation and the 045 orientation, $F(1, 720) = .004$, $p. < .951$.

The second planned comparison of the accuracy rates was conducted between the orientations of the 045 position and the 090 position. There was a significant difference between these two positions, $F(1, 720) = 10.02$, $p. < .001$. It was found that the participants were less accurate with the 090 orientations than the 045 degree position. This supports the main hypothesis in that the participants accuracy did decrease as the amount of mental rotation increased. However, this result was not anticipated by the third hypothesis, discussed later, which investigated the accuracy rates for the participants to mentally rotate and orientate the cardinal versus the non-cardinal headings.

The third planned comparison between the 090 orientation and the 135 degree orientation concluded that there was not a significant difference in the accuracy rates, $F(1, 720) = 1.79$, $p. < .181$. This comparison does not support the main hypothesis in that the participants answered the 135 degree orientation more accurately than the 090 degree orientation. This was not expected since the angle of rotation required was higher.

The planned comparison between the 135 orientation and the 180 orientation concluded that there was a significant difference in accuracy rates, $F(1, 720) = 6.54$, $p. < .011$. When the triangle was directly behind the aircraft symbol, at the 180 position, the participants were significantly more accurate at orientating the location of the triangle. This supports the main hypothesis since it was anticipated that the accuracy would become higher as the angle

of rotation approached the 180 position. As discussed and supported earlier, it was common for individuals to mentally rotate up to the 180 degree position. So even though it is numerically higher moving from the 090 to the 180 position, the actual amount of mental rotation becomes less when orientating with items from directly behind. Then, as the angle of rotation proceeds past the 180 position, the actual amount of required mental rotation begins to increase up to the 270 position. From that point, it begins to decrease again when approaching the 360 position, or at the straight ahead position. This is further supported by the almost symmetrical formation of the accuracy rates found on Figure 6.

The planned comparison of the accuracy between the 180 orientation and the 225 orientation did not support the hypothesis since there was a not a significant decrease in accuracy, $F(1, 720) = .626$, $p. < .429$. Again, there was the anticipated decrease in accuracy since the amount of required mental rotation was higher, but it was not significant.

The planned comparison conducted between the 225 orientation and the 270 concluded that there was a significant decrease in accuracy, $F(1, 720) = 13.35$, $p. < .001$. This comparison supports the main hypothesis since the increase in the amount of required mental rotation occurred with a decrease in accuracy.

The seventh planned comparison was conducted between the 270 orientation and the 315 orientation. The main hypothesis that the accuracy

rates would produce better results as the amount of required mental rotation decreased was again supported by the significant increase in accuracy, $F(1, 720) = 6.23$, $p. < .013$.

The planned comparison conducted between the 315 degree orientation and the 360 degree orientation was not significant for accuracy, $F(1, 720) = 1.63$, $p. < .201$. Similar to the 180 position, even though the numerical angle of rotation is higher, the actual amount of mental rotation is lower; thus producing higher rates of accuracy. The symmetrical formation of the mean accuracy rates correlating to the angles of required mental rotation can be easily identified when comparing the two 180 degree halves of Figure 6.

The final planned comparison for the response times was conducted between the cardinal headings/positions (360, 090, 180, & 270) and the non-cardinal headings (045, 135, 225, & 315). The fourth hypothesis which stated that the accuracy rates would be higher for the cardinal headings than for the non-cardinal headings was rejected since there was not a significant difference between them, $F(1, 720) = 2.22$, $p. < .136$.

The analysis of the interaction results for the response times and accuracy rates was not conducted due to its lack of relevance to the hypothesis and the overall scope of the study. The interactions would have been an investigation of the response times and accuracy rates over time; meaning, how they interacted and differed as the trials progressed

throughout the experiment. This would have been appropriate if the stimulus for each of the eight variables were identical. However, as previously stated in the design section, each participant completed a total of 64 trials, eight trials for each of the eight variables. The eight trials for each variable were not identical even though the correct responses were the same. The location and direction of the aircraft symbol appeared at all of the different headings possible at each of the eight 45 degree positions around the triangle. Therefore, with this specific experimental design, the investigation of the effects of the response times and accuracy rates over time would not be of great relevance to the testing of the hypothesis for this study.

Summary

The investigation of the relationships between the amount of mental rotation required for orientation, response times, and accuracy rates was conducted and three of the four hypothesis were supported by the statistical data analysis. The first hypothesis stated that as the amount of mental rotation required increased from the straight ahead position (360) and from the directly behind position (180), the response times will similarly increase. The overall ANOVA for this hypothesis concluded with a significant difference in response times between the eight 45 degree orientations. Additionally, all but one of the eight planned comparisons conducted between the eight orientations confirmed the hypothesis, as indicated by the "M" shaped curve in figure 5. However, even though the second planned comparison did not indicate a significant difference, the participants took longer to mentally rotate at the 045 degree position than at the 090 degree position. When compared to the results of the Hintzman, O'Dell, and Arndt (1981) study which provided the experimental design basis for this study, the curves depicting the function between response times and angular displacement are very similar (Figures 4 & 5).

Similarly, the third hypothesis which investigated the mental rotation of the cardinal directions (360, 090, 180, and 270) versus the non-cardinal directions (045, 135, 225, and 315) was tested. The hypothesis was confirmed with the participants taking significantly longer to respond to the

non-cardinal headings. Therefore, the participants took longer to orientate the position of the triangle as the amount of required mental rotation increased from their straight ahead and directly behind positions. This result supports the previously referenced studies such as that of Loftus (1978) and Hintzman et al. (1981).

These results along with those of Aretz (1988, 1989) suggests the design of future displays and interfaces should be a track-up design as well as providing the availability of rotating the display by 90 degrees. This feature may be helpful when using a static computer display such as a MRI or X-Ray. The image can be rotated by the four 90 degree positions (i.e., cardinal headings) to allow faster, maybe easier orientation and understanding of the items being displayed.

The second hypothesis stated that as the amount of mental rotation required increases from the straight ahead position (360) and from the directly behind position (180), the accuracy rates will decrease. The overall ANOVA for this hypothesis indicated a significant difference in accuracy rates between the eight 45 degree orientations. However, the results from this portion of the study were very surprising. Only four out of the eight planned comparisons confirmed the hypothesis with significant differences. Even though the high accuracy rates did occur as expected at the 360 and 180 orientations, the 090 and 270 orientations had the lowest accuracy rates. The resulting "W" shaped curve (figure 6) was anticipated, but not with the 090 and 270 orientations

resulting in lower accuracy rates than the two non-cardinal headings on either side of them. In other words, the "true" anticipated function (i.e., the Hintzman, et al. study, Figure 4) would have shown a decrease in accuracy between the 360 and 180 positions as well as the 090 and 270 positions resulting with higher accuracy rates than the 045, 135, 225, and 315 positions surrounding them respectively.

However, the results indicated that the participants had the hardest time locating whether the triangle was to the left or to the right of the aircraft symbol. Additionally, the majority of the incorrect responses made within these two orientations were of an inverse nature; meaning that the majority of the incorrect responses for the 090 orientation were answered as a 270 orientation and vice versa. This suggests that since the left and right decisions at the 180 degree orientation are reversed in relation to their position at the 360 degree orientation, the ability to accurately mentally rotate and orientate the 090 and 270 positions may be influenced by a possible reversal error. The ability to handle this reversal may be important.

Another possibility for the reversal errors of the two orientations may have been due to the influence of the target-centered experimental design. Unlike previous studies (i.e., Hintzman et al.), the stimulus of the experiment, the aircraft symbol, was not fixed while the target (the triangle) remained in the center of the screen. Therefore, the reversal problem of the subjects to accurately orientate the left and right positions may have also been due to

this change of stimulus and task.; thus possibly requiring a different cognitive process of conducting mental rotation required for spatial orientation.

As a result of this occurrence at the 090 and 270 orientations, the fourth hypothesis which investigated the accuracy rates for the mental rotation of the cardinal directions (360, 090, 180, and 270) versus the non-cardinal directions (045, 135, 225, and 315) was rejected. Even though there was not a significant difference between them, the cardinal headings did not score as highly as the non-cardinal orientations which was most likely due to the 090 and 270 phenomena. Additionally, other possible factors which may have been of influence for this occurance may be: 1) the location and distance differences of the response keys and 2) an ergonomically defined position of the hand used to respond was not specified.

Therefore, further research should explore the relationship between the amount of mental rotation required for spatial orientation and accuracy. Special attention should be applied to evaluating the conditions that lead to left and right reversal errors and their potential significance in flight. If such research will help to provide a better understanding between these two variables, then the design of future navigational displays and interfaces will perhaps result in more accurate performance by the users.

References

Aretz, A. J. (1988). A model of electronic map interpretation. *Proceedings of the Human Factors Society - 32nd Annual Meeting*, 130-134.

Aretz, A. J. (1989). Spatial cognition and navigation. *Proceedings of the Human Factors Society - 33rd Annual Meeting*, 8-12.

Berg, C., Hertzog, C., & Hunt, E. (1982). Age differences in the speed of mental rotation. *Developmental Psychology, 18*, 95-107.

Bethell-Fox, C., & Shephard, R. (1988). Mental rotation: Effects of stimulus complexity and familiarity. *Journal of Experimental Psychology: Human Perception and Performance, 14*(1), 12-23.

Burton, L. A., Wagner, N., Lim, C., & Levy, J. (1992). Visual field differences for clockwise and counterclockwise mental rotation. *Brain and Cognition, 18*, 192-207.

Cook, N. D., Fruh, H., Mehr, A., Regard, M., & Landis, T. (1994). Hemispheric cooperation in visuospatial rotations: Evidence for a manipulation role for the left hemisphere and a reference role for the right hemisphere. *Brain and Cognition, 25*, 240-249.

Cooper, L. A., & Podgorny, P (1976). Mental transformations and visual comparison processes: effects of complexity and similarity. *Journal of Experimental Psychology: Human Perception and Performance, 2*(4), 503-514.

Cooper, L. A., & Shepard, R. A. (1973). The time required to prepare for a rotated stimulus. *Memory and Cognition, 1*(3), 246-250.

Corballis, M. C., & Cullen, S. (1986). Decisions about the axes of disoriented shapes. *Memory and Cognition, 14*(1), 27-38.

Damos, D. A. (1991). Training mental rotation skills. In E. Farmer (Ed.), *Human Resource Management in Aviation* (pp. 67-70). Aldershot, England: Avebury Technical.

Hintzman, D. L., O'Dell, C. S., & Arndt, D. R. (1981). Orientation in cognitive maps. *Cognitive Psychology, 13*, 149-206.

Hock, H. S., & Ross, K. (1975). The effect of familiarity on rotational transformation. *Perception & Psychophysics, 8*(1), 15-20.

Hock, H. S., & Tromley, C. L. (1978). Mental rotation and perceptual uprightness. *Perception & Psychopyhsics, 24*(6), 529-533.

Jolicoeur, P. (1985).The time to name disorientated natural objects. *Memory & Cognition, 13*(4), 289-303.

Jones, B., & Anuza, T. (1982). Effects of sex, handedness, stimulus and visual field on "mental rotation". *Cortex, 18*, 501-514.

Koriat, A., & Norman, J. (1984). What is rotated in mental rotation? *Journal of Experimental Psychology: Learning, Memory, and Cognition, 10*(3), 421-434.

Koriat, A., & Norman, J. (1988). Frames and Images: Sequential effects in mental rotation. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 14*(1), 93-111.

Kozlowski, L. T., & Bryant, K. J. (1977). Sense of direction, spatial orientation, and cognitive maps. *Journal of Experimental Psychology: Human Perception and Performance, 3*(4), 590-598.

Loftus, G. R. (1978). Comprehending compass directions. *Memory & Cognition, 6*(4), 416-422.

Maki, R. H. (1986). Naming and locating the tops of rotated pictures. *Canadian Journal of Psychology, 40*(4), 368-387.

Pylyshyn, Z. W. (1979). The rate of "mental rotation" of images: A test of a holistic analogue hypothesis. *Memory & Cognition, 7*(1), 19-28.

Shepard, R. N., & Hurwitz, S. (1984). Upward direction, mental rotation, and discrimination of left and right turns in maps. *Cognition, 18*, 161-193.

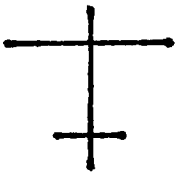
Shepard, R. N., & Metzler, J. (1971). Mental rotation of three-dimensional objects. *Science, 171*, 701-703.

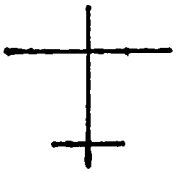
Thorndyke, P. W., & Hayes-Roth, B. (1982). Differences in spatial knowledge acquired from maps and navigation. *Cognitive Psychology, 14*, 560-589.

Uecker, A., & Obrzut, J. E. (1993). Hemisphere and gender differences in mental rotation. *Brain and Control, 22*, 42-50.

APPENDIX A

KEYBOARD LEGEND

7	8	9
4		6
1	2	3

315°	0°	45°
270°		90°
225°	180°	135°

APPENDIX B

MENTAL ROTATION/ORIENTATION COMPUTER PROGRAM

```
/* ---------------------------------------------------------
   Name          BOX C
   Type          Toolbox module
   Language      Microsoft QuickC version 2
   Video         Color or monochrome text mode
   ---------------------------------------------------------
*/

#include <stdio h>
#include <stdlib h>
#include <malloc h>
#include <dos h>
#include <string h>
#include <graph h>
#include "box h"

static void determine_video ( void ),
static unsigned video_seg = 0,
static char far *videoptr;
static int columns,

/* ---------------------------------------------------------
   Function      box_get()
   Toolbox       BOX C
   Demonstrated  BOXTEST C MENU C

   Parameters
    (input)     row1    Upper left corner of box
    (input)     col1    Upper left corner of box
    (input)     row2    Lower right corner of box
    (input)     col2    Lower right corner of box

   Returned      Address of far integer buffer containing data
                 saved from the rectangular area of screen

   Variables     i       Looping index for lines in box
                 width   Width of box area
                 height  Height of box area
                 bytes   Total number bytes to store box data
                 buf     Address of far buffer for storage
                 bufptr  Index into storage buffer memory
                 video_off Offset of video address for box data

   Description   Saves contents of a rectangular area of the
                 screen in a dynamically allocated buffer
   ---------------------------------------------------------
*/

unsigned far *box_get( unsigned row1, unsigned col1,
                       unsigned row2, unsigned col2 )
{
   unsigned i, width, height, bytes,
   unsigned far *buf, far *bufptr,
   unsigned video_off,

   /* Calculate the dimensions in bytes */
   width = ( col2 - col1 + 1 ) * 2,
   height = row2 - row1 + 1,
   bytes = height * width + 8

   /* Allocate storage space */
   if (( buf = (unsigned far *)malloc((size_t)bytes )) == NULL)
      {
      printf( "box_get()  malloc() failed\n" ),
      exit( 0 ),
      }

   /* Save the box coordinates in the buffer */
   bufptr = buf,
```

```
    *bufptr++ = row1,
    *bufptr++ = col1,
    *bufptr++ = row2,
    *bufptr++ = col2,

    /* Determine the text mode video segment and number of columns */
    determine_video(),

    /* Calculate starting location in video memory */
    video_off = (unsigned)(( columns * ( row1 - 1 ) +
            ( col1 - 1 )) * 2 ),

    /* Grab each line of the video */
    for ( i = 0, i < height, i++ )
       {
       movedata( video_seg, video_off,
              FP_SEG( bufptr ), FP_OFF( bufptr ), width ),
       bufptr += width / 2,
       video_off += columns * 2,
       }

    /* Return the buffer */
    return ( buf ),
}


/* ---------------------------------------------------------
   Function       box_put()
   Toolbox        BOX C
   Demonstrated   BOXTEST C MENU C

   Parameters
    (input)       buf      Far integer buffer previously created
                           by the function box_get()

   Returned       (function returns nothing)

   Variables      row1     Upper left corner of box
                  col1     Upper left corner of box
                  row2     Lower right corner of box
                  col2     Lower right corner of box
                  i        Loop index for each line of the box
                  width    Width of the box
                  height   Height of the box
                  bytes    Total number of bytes in the box
                  video_off Offset of video address for box data
                  workbuf  Index into the buffer

   Description    Restores screen contents that were saved in a
                  buffer by a previous call to box_get()
   ---------------------------------------------------------
*/

void box_put( unsigned far * buf )
{
    unsigned row1, col1, row2, col2,
    unsigned i, width, height, bytes,
    unsigned video_off,
    unsigned far *workbuf,

    /* Get the box coordinates */
    workbuf = buf,
    row1 = *workbuf++,
    col1 = *workbuf++,
    row2 = *workbuf++,
    col2 = *workbuf++,

    /* Calculate the dimensions in bytes */
    width = ( col2 - col1 + 1 ) * 2,
```

```
    height = row2 - row1 + 1;
    bytes = height * width;

    /* Determine the text mode video segment and number of columns */
    determine_video();

    /* Calculate starting location in video memory */
    video_off = ( columns * ( row1 - 1 ) + ( col1 - 1 )) * 2;

    /* Put each line out to video */
    for ( i = 0; i < height; i++ )
        {
        movedata( FP_SEG( workbuf ), FP_OFF( workbuf ),
                video_seg, video_off, width );
        workbuf += width / 2;
        video_off += columns * 2;
        }
}


/* ----------------------------------------------------------
   Function:     box_color()
   Toolbox:      BOX.C
   Demonstrated: BOXTEST.C MENU.C

   Parameters:
     (input)     row1    Upper left corner of box
     (input)     col1    Upper left corner of box
     (input)     row2    Lower right corner of box
     (input)     col2    Lower right corner of box

   Returned:     (function returns nothing)

   Variables:    x       Looping index for each row of box
                 y       Looping index for each column of box
                 fore    Current foreground text color
                 back    Current background text color
                 attr    Attribute byte combining fore and back

   Description:  Sets the foreground and background colors for
                 all characters in a box to the current colors.
                 Characters in the box are unaffected
   ----------------------------------------------------------
*/

void box_color ( unsigned row1, unsigned col1,
                 unsigned row2, unsigned col2 )
{
    unsigned x, y;
    unsigned fore;
    unsigned long back;
    unsigned char attr;

    /* Determine the text mode video segment and number of columns */
    determine_video();

    /* Build the attribute byte */
    fore = _gettextcolor();
    back = _getbkcolor();
    attr = (unsigned char)(( fore & 0xF ) |
        ((( fore & 0x10 ) >> 1 ) | back ) << 4 ));

    /* Work through the box */
    for ( x = row1 - 1; x < row2; x++ )
        for ( y = col1 - 1; y < col2; y++ )
            *( videoptr + ( columns * x + y ) * 2 + 1 ) = attr;
}
```

```
/* ---------------------------------------------------
   Function      box_charfill()
   Toolbox       BOX C
   Demonstrated  BOXTEST C MENUTEST C

   Parameters
     (input)     row1     Upper left corner of box
     (input)     col1     Upper left corner of box
     (input)     row2     Lower right corner of box
     (input)     col2     Lower right corner of box
     (input)     c        Character used to fill the box

   Returned      (function returns nothing)

   Variables     x        Looping index for each row of box
                 y        Looping index for each column of box

   Description   Fills a rectangular area of the screen with a
                 character  Attributes are unaffected
   ---------------------------------------------------
*/

void box_charfill ( unsigned row1, unsigned col1,
                                          unsigned row2, unsigned col2, unsigned char c )
{
   unsigned x, y,

   /* Determine the text mode video segment and number of columns */
   determine_video(),

   /* Work through the box */
   for ( x = row1 - 1, x < row2, x++ )
      for ( y = col1 - 1, y < col2, y++ )
         *( videoptr + ( columns * x + y ) * 2 ) = c,
}


/* ---------------------------------------------------
   Function      box_draw()
   Toolbox       BOX C
   Demonstrated  BOXTEST C MENU C

   Parameters
     (input)     row1     upper left corner of box
     (input)     col1     upper left corner of box
     (input)     row2     lower right corner of box
     (input)     col2     lower right corner of box
     (input)     line_type Indicates single-line or double-
                    line box border (or none)

   Returned      (function returns nothing)

   Variables     x        Keeps track of horizontal position
                 y        Keeps track of vertical position
                 dx       Horizontal motion increment
                 dy       Vertical motion increment
                 c        Character for each part of the border

   Description   Draws a single-line or double-line box border
                 around a box  Does not affect attributes
   ---------------------------------------------------
*/

void box_draw( unsigned row1, unsigned col1,
          unsigned row2, unsigned col2, unsigned line_type )
{
   unsigned x, y, dx, dy,
   unsigned c,
```

```
/* Determine the text mode video segment and number of columns */
determine_video(),

/* Work around the box */
x = col1,
y = row1,
dx = 1,
dy = 0,
do
  {

  /* Set the default character for unbordered boxes */
  c = ' ',

  /* Set the single-line drawing character */
  if ( line_type == 1 )
    if ( dx )
      c = 196,
    else
      c = 179,

  /* Set the double-line drawing character */
  else if ( line_type == 2 )
    if ( dx )
      c = 205,
    else
      c = 186,

  /* Change direction at top right corner */
  if ( dx == 1 && x == col2 )
    {
    dx = 0,
    dy = 1,
    if ( line_type == 1 )
      c = 191,
    else if ( line_type == 2 )
      c = 187,
    }

  /* Change direction at bottom right corner */
  if ( dy == 1 && y == row2 )
    {
    dx = -1,
    dy = 0,
    if ( line_type == 1 )
      c = 217,
    else if ( line_type == 2 )
      c = 188,
    }

  /* Change direction at bottom left corner */
  if ( dx == -1 && x == col1 )
    {
    dx = 0,
    dy = -1,
    if ( line_type == 1 )
      c = 192,
    else if ( line_type == 2 )
      c = 200,
    }

  /* Check for top left corner */
  if ( dy == -1 && y == row1 )
    {
    if ( line_type == 1 )
      c = 218,
    else if ( line_type == 2 )
      c = 201,
    }
```

```
    /* Put new character to video */
    *( videoptr + ( columns * ( y - 1 ) + ( x - 1 )) * 2 ) = (char)c,

    /* Move to next position */
    x += dx,
    y += dy,

    }
  while ( dy != -1 || y >= row1 ),
}


/* ---------------------------------------------------
  Function      box_erase()
  Toolbox       BOX C
  Demonstrated  BOXTEST C MENU C

  Parameters
   (input)      row1    Upper left corner of box
   (input)      col1    Upper left corner of box
   (input)      row2    Lower right corner of box
   (input)      col2    Lower right corner of box

  Returned      (function returns nothing)

  Variables     i       Looping index for each row of the box
                buf     String of spaces for each row

  Description   Fills a box with spaces  Uses the current color
                attributes
  ---------------------------------------------------
*/

void box_erase( unsigned row1, unsigned col1,
          unsigned row2, unsigned col2 )
{
  unsigned i,
  char buf[81],

  /* Fill the buffer with spaces */
  sprintf( buf, "%*s", col2 - col1 + 1, "" ),

  /* Put each line out to video */
  for ( i = row1, i <= row2, i++ )
    {
    _settextposition( i, col1 ),
    _outtext( buf ),
    }
}


/* ---------------------------------------------------
  Function      determine_video()
  Note          STATIC FUNCTION AVAILABLE ONLY TO THIS MODULE
  Language      Microsoft QuickC
  Toolbox       BOX C

  Parameters    (none)

  Returned      (function returns nothing)

  Variables     (none)

  Description   Determines the text mode video segment and the
                number of character columns currently set
                Fills in static variables that are
    available only to the functions in this module
  ---------------------------------------------------
```

```
*/

static void determine_video( void )
{
   if ( !video_seg )
      {
      /* Determine the text mode video segment */
  switch ( *((char far *)0x449) )
        {
        case 0:
        case 1:
        case 2:
        case 3:
           video_seg = 0xB800;
           videoptr = (char far *)0xB8000000;
           break;
        case 7:
           video_seg = 0xB000;
           videoptr = (char far *)0xB0000000;
           break;
        default:
           printf( "BOX.C: not in text mode\n" );
           exit( 0 );
        }

      /* Determine number of columns for current text mode */
  columns = *( (int far *)0x44A );
   }
}
```

```
/* ---------------------------------------------------
   Name          DATA_PLT C
   Type          Student data routines
                 Air Traffic Control Screening Program
   Language      Microsoft QuickC version 2
   ---------------------------------------------------
*/

#include <stdio h>
#include "getkey h"
#include "typ_init h"
#include "edit h"
#include "list h"
#include "file h"
#include "menu h"
#include "box h"
#include "data_plt h"
#include "t_colors h"

#define right  "RIGHT"
#define left   "LEFT"
#define male   "MALE"
#define female "FEMALE"

char *info_box_1[] =
   {
   " Student Information Entry ",
   " Have you already been entered into the ",
   " roster of qualified users? ",
   "< Yes or No >",
   NULL
   },

char *info_box_2[] =
   {
   " Student Information Entry ",
   "",
   " Enter a unique 9 character identifier that ",
   " I can use to identify you in the future  ",
   " Most people use their SS# number  ",
   "",
   "     > ",
   "<>",
   NULL
   },

char *info_box_3[] =
   {
   " Student Information Entry ",
   "",
   " Please enter your unique identifier at the ",
   " identifier below  ",
   "",
   "     > ",
   "<>",
   NULL
   },

char *info_box_4[] =
   {
   " Student Information Entry ',
   "",
   " Are you RIGHT handed or are you LEFT handed? ",
   "",
   "< Right or Left >",
   NULL
   },

char *info_box_5[] =
```

```
  {
  " Student Information Entry ",
  "",
  " Is it correct that you are RIGHT handed ? ",
  "",
  "< Yes or No >",
  NULL
  },

char *info_box_6[] =
  {
  " Student Information Entry ",
  "",
  " Is it correct that you are LEFT handed ? ",
  "",
  "< Yes or No >",
  NULL
  },

char *info_box_7[] =
  {
  " Student Information Entry ",
  "",
  " I'm sorry you are not in the table ",
  " of registered users  ",
  "",
  "< Press any key >",
  NULL
  },

char *info_box_8[] =
  {
  " Student Information Record ",
  "",
  " Are you MALE or FEMALE ? ",
  "",
  "< Male or Female >",
  NULL
  },

char *info_box_9[] =
  {
  " Student Information Record ",
  "",
  " Student Identifier              ",
  "",
  " Right or Left handed  ",
  "",
  " Male or Female        ",
  "",
  " IS THE ABOVE INFORMATION CORRECT? ",
  "",
  "< Yes or No >",
  NULL
  },

char *info_box_10[] =
  {
  " Student Information Entry ",
  "",
  " Is it correct that you are a MALE ? ",
  "",
  "< Yes or No >",
  NULL
  },

char *info_box_11[] =
  {
  " Student Information Entry ",
```

```
    "",
    " Is it correct that you are a FEMALE ? ",
    "",
    "< Yes or No >",
    NULL
    },

  char *info_box_20[] =
    {
    " Student Information Entry ",
    "",
    " I'm sorry another user already uses that ",
    " identifier  Please try another one  ",
    "",
    "< Press any key >",
    NULL
    },

  char *drop_right_left[] =
    {
    "",
    "Right",
    "Left",
    "",
    NULL
    },

  char *drop_male_female[] =
  {
    "",
    "Male",
    "Female",
    "",
    NULL
    },

  char *drop_yes_no[] =
    {
    "",
    "Yes",
    "No",
    "",
    NULL
    },

  /* ---------------------------------
    Function       Function to determine whether candidate
                   is male or female

    File           TEST_1 C

    Parameters     None

    Returned
     (output)      'M' - if candidate is male
                   'F' - if candidate is female

    Variables      None

    Description    Function to determine whether candidate is male
                   or female
    ---------------------------------
  */

  char Male_or_female( void )
  {
    int finish = 0,
    int male_female,
    int *save_info_box,
```

```
  int yes_no,

  while( finish == 0 ) {
    /* Display info_box_8 */
    save_info_box = menu_message( 5, 8, info_box_8 ),

    /* Get student answer male or female ? */
    menu_erase( menu_drop( 12, 30, drop_male_female, &male_female )),

    /* Erase info_box_8 */
    menu_erase( save_info_box ),

    /*
      male_female = 1 ==> Male
      male_feamle = 2 ==> Female
    */

    if ( male_female == 1 ) {
      /* Confirm whether student is male */
      /* Display info_box_10 */
      save_info_box = menu_message( 5, 8, info_box_10 ),

      /* Get student answer yes or no ? */
      menu_erase( menu_drop( 12, 30, drop_yes_no, &yes_no )),

      /* Erase info_box_10 */
      menu_erase( save_info_box ),
      }
    else {
      /* Confirm whether student is female */
      /* Display info_box_11 */
      save_info_box = menu_message( 5, 8, info_box_11 ),

      /* Get student answer yes or no ? */
      menu_erase( menu_drop( 12, 30, drop_yes_no, &yes_no )),

      /* Erase info_box_11 */
      menu_erase( save_info_box ),
      }
    if ( yes_no == 1 )
      /* Student entry was correct => set flag to quit loop */
      finish = 1,
  }
  if ( male_female == 1 )
    return('M'),
  else
    return('F'),
}


/*
  Function to determine whether student is right
  or left handed
*/

char Right_or_left_handed(void)
{
  int finish = 0,
  int right_or_left,
  int yes_no,
  int *save_info_box,

  while( finish == 0 ) {
    /* Display info_box_4 */
    save_info_box = menu_message( 5, 8, info_box_4 ),

    /* Get student answer right or left ? */
    menu_erase( menu_drop( 12, 30, drop_right_left, &right_or_left )),
```

```
    /* Erase info_box_4 */
    menu_erase( save_info_box );

    /*
      right_or_left = 1 ==> Right handed
      right_or_left = 2 ==> Left handed
    */

    if ( right_or_left == 1 ) {
      /* Confirm whether student is right handed */
      /* Display info_box_5 */
      save_info_box = menu_message( 5, 8, info_box_5 );

      /* Get student answer yes or no ? */
      menu_erase( menu_drop( 12, 30, drop_yes_no, &yes_no ));

      /* Erase info_box_5 */
      menu_erase( save_info_box );
      }
    else {
      /* Confirm whether student is left handed */
      /* Display info_box_6 */
      save_info_box = menu_message( 5, 8, info_box_6 );

      /* Get student answer yes or no ? */
      menu_erase( menu_drop( 12, 30, drop_yes_no, &yes_no ));

      /* Erase info_box_6 */
      menu_erase( save_info_box );
      }

    if ( yes_no == 1 )

      /* Student entry was correct => set flag to quit loop */
      finish = 1;

  }
  if ( right_or_left == 1 )
    return('R');
  else
    return('L');
}


/*
  Define procedure for getting student data plate
*/

void get_student_data( NODE *h, STUDENT_RECORD *new_student, long *positn )
{
  int counter;
  int qualified_user_answer;
  int far *save_info_box;
  int finish = 0;
  int key;
  long offset;
  char unique_ident[10];
  char r_l_handed;
  char male_female;

  while ( finish == 0 ) {
    /* Clear unique_ident */
    for ( counter = 0; counter <= 8; counter++ )
      unique_ident[counter] = ' ';
    unique_ident[9] = '\0';

    /* Display info_box_1 */
    save_info_box = menu_message( 5, 8, info_box_1 );
```

```
/* Get student answer yes or no ? */
menu_erase( menu_drop( 10, 30, drop_yes_no, &qualified_user_answer )),

/* Erase info_box_1 */
menu_erase( save_info_box ),

/*
  qualified_user_answer = 1 ==> yes
  qualified_user_answer = 2 ==> no
*/

if ( qualified_user_answer == 1 ) {
  /* Display info_box_3 */
  save_info_box = menu_message( 5, 8, info_box_3 ),

  /* Get unique identifier */
  _settextposition( 10, 18 ),
  editline( unique_ident ),

  /* Erase info_box_3 */
  menu_erase( save_info_box ),

  /* Check student identifier with those held on disk */
  offset = check( h, unique_ident ),

  /* Save position on disk */
  *positn = offset,

  /*
    offset == 0   => student not registered
    offset <> 0   => student is a registered user
  */

  if ( offset == 0L ) {
    /* Display info_box_7 */
    save_info_box = menu_message( 10, 8, info_box_7 ),

    getkey_or_mouse(),

    /* Erase info_box_7 */
    menu_erase( save_info_box ),
    }
  else {
    /* fetch student record */
    Fetch( offset, new_student ),

    /* Set finish flag to 1 */
    finish = 1,
    }
  }
else {
  /* Display info_box_2 */
  save_info_box = menu_message( 5, 8, info_box_2 ),

  /* Get unique identifier */
  _settextposition( 11, 18 ),
  editline( unique_ident ),

  /* Erase info_box_2 */
  menu_erase( save_info_box ),

  /* Check student identifier with those held on disk */
  offset = check( h, unique_ident ),

                    /* Save position on disk */
                    *positn = offset,

                    /* offset <> 0L then another student uses that identifier */
  if ( offset != 0L ) {
```

```
        /* Display info_box_20 */
        save_info_box = menu_message( 10, 8, info_box_20 );

        getkey_or_mouse();

        /* Erase info_box_20 */
        menu_erase( save_info_box );
        }
      else {
        /* Determine whether right or left handed */
        r_l_handed = Right_or_left_handed();

        /* Determine whether student is male or female */
        male_female = Male_or_female();

        /* Last chance for student to validate entered information */
        /* Display info_box_9 */
        save_info_box = menu_message( 3, 8, info_box_9 );

        /* Display student identifier */
        _settextposition( 5, 33 );
        _outtext( unique_ident );

        /* Display whether student is right or left handed */
        _settextposition( 7, 33 );
        if ( r_l_handed == 'R' )
          _outtext( right );
        else
          _outtext( left );

        /* Display whether student is male or female */
        _settextposition( 9, 33 );
        if ( male_female == 'M' )
          _outtext( male );
        else
          _outtext( female );

        /* Get student answer yes or no ? */
        menu_erase( menu_drop( 16, 30, drop_yes_no, &qualified_user_answer ));

        /* Entered information correct */
        if ( qualified_user_answer == 1 ) {

          /* Initialize student record for new student */
          for ( counter = 0; counter <= 9; counter++ )
            new_student->qualifier[counter] = unique_ident[counter];
          new_student->r_l_handed = r_l_handed;
          new_student->male_female = male_female;

          /* Set finish flag to 1 */
          finish = 1;
        }

      /* Erase info_box_9 */
      menu_erase( save_info_box );
      }
    }
  }
}
```

```
/* ---------------------------------------------------
   Name         DSK_INIT C
   Type         Routines to manipulate student data, and
                student index files on disk
                Air Traffic Control Screening Program
   Language     Microsoft QuickC version 2
   ---------------------------------------------------
*/

#include <stdio h>
#include <conio h>
#include "getkey h"
#include "typ_init h"
#include "list h"
#include "file h"
#include "dsk_init h"
#include "menu h"
#include "box h"
#include "t_colors h"


/* Error message data */
char *error_box_1_01[] =
  {
  " Error Message #1 01 ",
  "",
  " Unable to access the following ",
  " file  STUDENT DAT ",
  "",
  "< Press any key >",
  NULL
  },

char *error_box_1_02[] =
  {
  " Error Message #1 02 ",
  "",
  " Unable to access the following ",
  " file  STUDENT NDX ",
  "",
  "< Press any key >",
  NULL
  },

/* ---------------------------------
   Function      Initialize()
   File          DSK_INIT C

   Parameters
    (input)      hd     pointer to head of linked list of type
                        NODE
                 tl     pointer to tail of linked list of type
                        NODE

   Returned      (function returns nothing)

   Variables     result   Return value from function call
                          1 = file on disk
                          0 = file not on disk
                 nrecs    Number of records in student data file

   Description   Function to detrmine whether student index file
                 is on disk  If there exists a student index file
                 contents of it are read into a linked list
   ---------------------------------
*/

void Initialize( NODE **hd, NODE **tl )
{
```

```
int result;
int nrecs;
int counter;
int *save_error_box;

        /* set the head and tail pointers */
        *hd = NULL; *tl = NULL;

        /*
  Create index file
*/
Create_index_file();

/* Determine whether index file is on disk */
result = Index_on_disk();
/*
  result == 1   =>   Index on disk
  result == 0   =>   Index not on disk
*/
if ( result == 0 )
  /*
    Index not on disk return to caller.
  */
  return;
else {
  /*
    Read student data file to determine number of records in file.
  */
  nrecs = Num_records();
  if ( nrecs == 0 ) {
    /*
      nrecs == 0  =>   Error reading student data file
    */

    /*
      set error box color to red
      set error text color to white
    */
    menu_back_color( BK_RED );
    menu_text_color( T_WHITE | T_BRIGHT );

    /* Display error_box_1_01 */
    save_error_box = menu_message( 10, 8, error_box_1_01 );

    getkey_or_mouse();

    /* Erase error_box_1_01 */
    menu_erase( save_error_box );

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE );
    menu_text_color( T_BLACK );
    }
  else {
    /*
      Read information in indexfile
      into linked list
    */
    result = Index_to_link_list( nrecs, hd, tl );
    /*
      result == 1   =>   Function successful
      result == 0   =>   Error in reading file!
    */
    if ( result == 0 ) {
      /*
        set error box color to red
```

```
          set error text color to white
        */
        menu_back_color( BK_RED ),
        menu_text_color( T_WHITE | T_BRIGHT ),

        /* Display error_box_1_02 */
        save_error_box = menu_message( 10, 8, error_box_1_02 ),

        getch(),

        /* Erase error_box_1_02 */
        menu_erase( save_error_box ),

        /*
          set box color back to cyan
          set text color back to black
        */
        menu_back_color( BK_WHITE ),
        menu_text_color( T_BLACK ),
      }
    }
  }
}


/* ---------------------------------
  Function      Stats_initialize()
  File          DSK_INIT C

  Parameters
   (input)      hd     pointer to head of linked list of type
                       RES_NODE
                tl     pointer to tail of linked list of type
                       RES_NODE

  Returned      (function returns nothing)

  Variables     result   Return value from function call
                         1 = file on disk
                         0 = file not on disk
                nrecs    Number of records in student data file

  Description   Function to detrmine whether student data file
                is on disk  If there exists a student data file
                contents of it are read into a linked list
  ---------------------------------
*/

void Stats_initialize( RES_NODE **hd, RES_NODE **tl )
{
  int result,
  int nrecs,
  int *save_error_box,

  /* Determine whether student data file is on disk */
  result = File_on_disk(),

  /*
    result == 1   =>   File on disk
    result == 0   =>   File not on disk
  */
  if ( result == 0 )
    /*
      Index not on disk return to caller
    */
    return,
  else {
    /*
      Read student data file to determine number of records in file
```

```
*/
nrecs = Num_records();
if ( nrecs == 0 ) {
  /*
    nrecs == 0  =>   Error reading student data file
  */

  /*
    set error box color to red
    set error text color to white
  */
  menu_back_color( BK_RED );
  menu_text_color( T_WHITE | T_BRIGHT );

  /* Display error_box_1_01 */
  save_error_box = menu_message( 10, 8, error_box_1_01 ),

  getkey_or_mouse();

  /* Erase error_box_1_01 */
  menu_erase( save_error_box ),

  /*
    set box color back to cyan
    set text color back to black
  */
  menu_back_color( BK_WHITE );
  menu_text_color( T_BLACK ),
  }
else {
  /*
    Read information in indexfile
    into linked list
  */
  result = Student_data_to_link_list( hd , tl );

  /*
    result == 1   =>   Function successful
    result == 0   =>   Error in reading file!
  */
  if ( result == 0 ) {
    /*
      set error box color to red
      set error text color to white
    */
    menu_back_color( BK_RED );
    menu_text_color( T_WHITE | T_BRIGHT ),

    /* Display error_box_1_01 */
    save_error_box = menu_message( 10, 8, error_box_1_01 ),

    getkey_or_mouse();

    /* Erase error_box_1_01 */
    menu_erase( save_error_box ),

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE ),
    menu_text_color( T_BLACK ),
```

```
/* ---------------------------------------------------
   Name          EDIT C
   Type          Toolbox module
   Language      Microsoft QuickC
   Demonstrated  EDITTEST C
   Video         (no special video requirements)
   ---------------------------------------------------
*/

#include <stdio h>
#include <stdlib h>
#include <conio h>
#include <string h>
#include <graph h>
#include "edit h"
#include "getkey h"

/* ---------------------------------
   Function      next_word()
   Toolbox       EDIT C
   Demonstrated  EDITTEST C

   Parameters
    (input)     str     String to be evaluated
    (input)     ndx     Character position

   Returned     Character position of next word

   Variables    len     Length of the string

   Description  Finds the start of the next word in the string
   ---------------------------------
*/

int next_word( char *str, int ndx )
{
   unsigned len,

   /* Get the length of the string */
   len = strlen( str ),

   /* Move to end of the current word */
   while ( ndx < len && str[ndx] != ' ' )
      ndx++,

   /* Move to the start of the next word */
   while ( ndx < len && str[ndx] == ' ' )
      ndx++,

   /* If at end of string, back up to start of last word */
   if ( ndx == len )
      {
      ndx--,

      /* Move back over any spaces */
      while ( ndx >= 0 && str[ndx] == ' ' )
         ndx--,

      /* Move back over preceding word */
      while ( ndx >= 0 && str[ndx] != ' ' )
         ndx--,

  /* Move one step forward to start of preceding word */
      ndx++,
      }

   /* Return the new position */
   return ( ndx ),
}
```

```
/* ---------------------------------
   Function      prev_word()
   Toolbox       EDIT C
   Demonstrated  EDITTEST C

   Parameters
    (input)     str     String to be evaluated
    (input)     ndx     Character position

   Returned     Character position of previous word

   Variables    len     Length of the string

   Description  Finds start of the previous word in the string
   ---------------------------------
*/

int prev_word( char *str, int ndx )
{
   int len,

   /* Get length of the string */
   len = strlen( str ),

   /* Move back over nonspace characters in current word */
   while ( ndx && str[ndx] != ' ' )
      ndx--,

   /* Move back over the spaces between words */
   while ( ndx && str[ndx] == ' ' )
      ndx--,

   /* Move back over characters in previous word */
   while ( ndx >= 0 && str[ndx] != ' ' )
      ndx--,

   /* Move to first character of the word */
   while (( ndx < len && str[ndx] == ' ' )  || ( ndx < 0 ))
      ndx++,

   /* If all spaces, then move back to start of string */
   if ( ndx == len )
      ndx = 0,

   /* Return the new position */
   return ( ndx ),
}


/* ---------------------------------
   Function      delete_char()
   Toolbox       EDIT C
   Demonstrated  EDITTEST C

   Parameters
    (input)     str     String to be evaluated
    (input)     ndx     Character position

   Returned     Character position

   Variables    (none)

   Description  Deletes one character from the string
   ---------------------------------
*/

int delete_char( char *str, int ndx )
```

```
{
   int ndx_start,

   /* Save current ndx */
   ndx_start = ndx,

   /* Shuffle characters back one space */
   while ( str[ndx] )
      {
      str[ndx] = str[ndx + 1],
      ndx++,
      }

   /* Return the unchanged position */
   return ( ndx ),
}


/* ---------------------------------
   Function      insert_char()
   Toolbox       EDIT C
   Demonstrated  EDITTEST C

   Parameters
    (input)     str     String to be evaluated
    (input)     ndx     Character position
    (input)     c       Character to be inserted

   Returned     Next character position

   Variables    i       Looping index

   Description  Inserts a character into the string
   ---------------------------------
*/

int insert_char( char *str, int ndx, char c )
{
   int i,

   /* Shuffle characters right one space */
   for ( i = strlen( str ) - 1, i > ndx, i-- )
      str[i] = str[i-1],

   /* Put character in new position */
   str[ndx] = c,

   /* Return next character position */
   return ( ++ndx ),
}


/* ---------------------------------
   Function      insert_spaces()
   Toolbox       EDIT C
   Demonstrated  EDITTEST C

   Parameters
    (input)     str     String to be evaluated
    (input)     ndx     Character position
    (input)     n       Number of spaces

   Returned     Next character position

   Variables    i       Looping index

   Description  Inserts a character into the string
   ---------------------------------
*/
```

```
int insert_spaces( char *str, int ndx, int n )
{
   int i,

   /* Shuffle characters to the right n places */
   for ( i = strlen( str ), i >= ndx, i-- )
      str[i + n] = str[i],

   /* Put n spaces in string */
   while ( n-- )
      str[++i] = ' ',

   /* Move to the first character after inserted spaces */
   return ( ndx + n - 1 ),
}


/* ---------------------------------
   Function       replace()
   Toolbox        EDIT C
   Demonstrated   EDITTEST C

   Parameters
    (input)      str      String to be evaluated
    (input)      substr1  Sub string to find
    (input)      substr2  Sub string to replace substr1

   Returned       Number of replacements made

   Variables      count    Count of replacements made
                  len      Length of str
                  len2     Length of substr2
                  i        Looping index
                  shift    Amount to shift for insert

   Description    Replaces each occurrence of substr1 in str
                  with substr2
   ---------------------------------
*/

int replace( char *str, char *substr1, char *substr2 )
{
   int count = 0,
   int len, len2,
   int i, shift,

   /* Get length of replacement string */
   len2 = strlen( substr2 ),

   /* Determine amount of shift for each replacement */
   shift = len2 - strlen( substr1 ),

   /* Process each occurrence of substr1 in str */
   while (( str = strstr( str, substr1 )) != NULL )
      {

      /* Keep track of number of replacements */
      count++,

      /* Find current length of str */
      len = strlen( str ),

      /* Shift left if substr2 is shorter than substr1 */
      if ( shift < 0 )
         {
         for ( i = abs( shift ), i < len + 1, i++ )
            str[i + shift] = str[i],
         }
```

```
      /* Shift right if substr2 is longer than substr1 */
      else if ( shift > 0 )
         {
         for ( i = len, i, i-- )
            str[i + shift] = str[i],
         }

      /* Copy substr2 into new place in str */
      strncpy( str, substr2, len2 ),

      /* Increment str pointer to character beyond replacement */
      str += len2,
      }

   /* Return the number of replacements made */
   return ( count ),
}


/* ---------------------------------
   Function       editline()
   Toolbox        EDIT C
   Demonstrated   EDITTEST C

   Parameters
    (input)      str       String to be edited

   Returned       KEY_UP        If Cursor Up was last keypress
       KEY_DOWN      If Cursor Down was last keypress
       KEY_ESCAPE    If Escape was last keypress
       KEY_ENTER     If Enter was last keypress

   Variables      doneflag      Signals when to end the edit
                  insertflag    Insert or overstrike mode
                  index         Cursor position
                  key           Key code returned by getkey()
                  len           Length of str
                  i             Looping index
                  strpos        Original cursor position

   Description    Displays string at the current cursor location,
                  uses the current text colors and allows user
                  to edit the string with standard editing keys
   ---------------------------------
*/

int editline( char *str )
{
   unsigned doneflag = 0,
   int insertflag = 1, index = 0,
   int key, len  i,
   struct rccoord strpos,

   /* Get the length of the string to be edited */
   len = strlen( str ),

   /* Record current location of the cursor */
   strpos = _gettextposition(),

   /* Clear out any keypresses in the keyboard buffer */
   while ( kbhit() )
     getch(),

   /* Main editing loop */
   while ( !doneflag )
      {

      /* Position the cursor at the original location */
```

```
    _settextposition( strpos row, strpos col ),

    /* Display the string */
    _outtext( str ),

    /* Move cursor to current editing position */
    _settextposition( strpos row, strpos col + index ),

    /* Set cursor type for insert or overstrike mode */
    if ( insertflag )
       _settextcursor( CURSOR_UNDERLINE ),
    else
       _settextcursor( CURSOR_BLOCK ),

    /* Wait for a keypress or mouse movement */
    key = getkey_or_mouse(),

/* Process each keypress */
    switch (key)
       {

       case KEY_UP
          doneflag = key,
          break,

       case KEY_DOWN
          doneflag = key,
          break,

       case KEY_LEFT
          if ( index )
             index--,
          break,

       case KEY_RIGHT
          if ( index < len - 1 )
             index++,
          break;

       case KEY_ESCAPE
          doneflag = key,
          break,

       case KEY_CTRL_LEFT
          index = prev_word( str, index ),
          break,

       case KEY_CTRL_RIGHT
          index = next_word( str, index ),
          break,

       case KEY_END
          for ( index = len - 1, str[index] == ' ' && index, index-- )
             {,}
          if ( index && index < len - 1 )
             index++,
          break,

       case KEY_BACKSPACE
          if ( index )
             {
             index--,
             delete_char( str, index ),
             str[len-1] = ' ',
             }
          break,

       case KEY_CTRL_END
          for ( i = index, i < len, i++ )
```

```
            str[i] = ' ';
         break;

      case KEY_INSERT:
         insertflag ^= 1;
         break;

      case KEY_DELETE:
         delete_char( str, index );
         str[len-1] = ' ';
         break;

      case KEY_ENTER:
         doneflag = key;
         break;

      case KEY_HOME:
         index = 0;
         break;

      default:
         if ( key >= ' ' && key < 256 )
            {
            if ( insertflag )
               insert_char( str, index, (char)key );
            else
               str[index] = (char)key;
            if ( index < len - 1 )
               index++;
            }
         break;
      }

   /* Truncate string at original length */
   str[len] = 0;
   }

/* Return the key that caused the exit */
return ( doneflag );
```

```
/* ---------------------------------------------------------
   Name          FILE C
   Type:         Disk file handling routines for
                 Air Traffic Control Screening Program
   Language      Microsoft QuickC version 2
   ---------------------------------------------------------
*/

#include <stdio h>
#include <string h>
#include <conio h>
#include "getkey h"
#include "typ_init h"
#include "list.h"
#include "file h"
#include "menu h"
#include "t_colors h"
#include "sound h"


/* Define error messages */
char *error_box_1_03[] =
   {
   " Error Message #1 03 ",
   "",
   " Attempt to reposition file pointer ",
   " in file  STUDENT DAT failed ",
   "",
   "< Press any key >",
   NULL
   },

char *error_box_1_04[] =
   {
   " Error Message #1 04 ",
   "",
   " There are no records in student ",
   " data file to read  ",
   " Unable to do statistical ",
   " analysis of student results  ",
   "",
   "< Press any key >",
   NULL
   },

char *error_box_1_05[] =
   {
   " Error Message #1 05 ",
   "",
   " Unable to save student record ",
   " in student data file  ",
   " ",
   " Result => the current student ",
   " does not have his record saved ",
   " on disk ",
   "",
   "< Press any key >",
   NULL
   },

char *error_box_1_06[] =
   {
   " Error Message #1 06 ",
   "",
   " Unable to update student record ",
   " in student data file  ",
   " ",
   " Result => the current student ",
   " record in the student data file ",
```

```
  " does not contain the latest test ",
  " results  ",
  "",
  "< Press any key >",
  NULL
  },

char *error_box_1_07[] =
  {
  " Error Message #1 07 ",
  "",
  " Unable to create student index ",
  " file from student data file  ",
  " ",
  " Result => the program will not ",
  " be able to access student records ",
  " held on disk  ",
  "",
  "< Press any key >",
  NULL
  },


/* ---------------------------------
  Function      Index_on_disk()
  File          FILE C

  Parameters    (none)

  Returned      1       Student index file is on the disk
                0       Student index file is not on disk

  Variables     check   file pointer to student data file


  Description   Function to determine whether the student
                index file is located in the current directory
  ---------------------------------
*/

int Index_on_disk( void )
{
  FILE *check,

  /* Attempt to open index file on disk */
  if (( check = fopen( INDEX, "rb" )) != NULL) {
    fclose( check ),                  /* Close disk file          */
    return( 1 ),                      /* File on disk => return 1  */
    }
  else
    return( 0 ),                      /* Not on disk => return 0   */
}


/* ---------------------------------
  Function      File_on_disk()
  File          FILE C

  Parameters    (none)

  Returned      1       Student data file is on the disk
                0       Student data file is not on disk

  Variables     check   file pointer to student data file


  Description   Function to determine whether the student
                data file is located in the current directory
  ---------------------------------
```

```
*/


int File_on_disk( void )
{
  FILE *check;

  /* Attempt to open index file on disk */
  if (( check = fopen( FILENAME, "rb" )) != NULL) {
    fclose( check );                  /* Close disk file          */
    return( 1 );                      /* File on disk => return 1   */
    }
  else
    return( 0 );                      /* Not on disk => return 0    */
}

/*
  Procedure to read student
  data file into linked list
  to allow manipulation for
  statistical analysis
*/

int Student_data_to_link_list( RES_NODE **h, RES_NODE **t )
{
  FILE *check;
  STUDENT_RECORD record;
  int *save_error_box;
  int result;
  int counter;
  int nrecs;

  /* Open index file on disk */
  if (( check = fopen( FILENAME, "rb" )) != NULL) {

    /* get number of records to read */
    nrecs = getw( check );

    /* if number of records <= 0 then error */
    if ( nrecs <= 0 ) {

      /*
        set error box color to red
        set error text color to white
      */
      menu_back_color( BK_RED );
      menu_text_color( T_WHITE | T_BRIGHT );

      /* Display error_box_1_04 */
      save_error_box = menu_message( 10, 8, error_box_1_04 );

      /* Error Sound */
      warble( 5 );

      /* Get key/mouse press from user */
      getkey_or_mouse();

      /* Erase error_box_1 */
      menu_erase( save_error_box );

      /*
        set box color back to cyan
        set text color back to black
      */
      menu_back_color( BK_WHITE );
      menu_text_color( T_BLACK );

      fclose( check );                /* Close disk file          */
      return( 0 );                    /* Read unsuccessful return 0 */
```

```
      }
    else {
      /* loop size defined by number of recs to read */
      for ( counter = 1, counter <= nrecs, counter++ ) {

        /* read index record from disk */
        fread( &record, sizeof( STUDENT_RECORD ), 1, check ),

        /* insert index record into linked list */
        res_addsl( &record, h, t ),
        }
      fclose( check ),              /* Close disk file        */
      return( 1 ),                  /* Read successful return 1  */
    }
  }
  else
    return( 0 ),                    /* File open failed! return 0 */
}


/* ---------------------------------
  Function      Write_num_records(),
  File          FILE C

  Parameters
   (input)      number_records    value to insert into the number
                                  of records field in the student
                                  file

  Returned      integer           1 = successfull write
                                  0 = failure

  Variables     random            logical name for student data file
                nrecs             number of records in student data
                                  file
                counter           loop counter

  Description   Inserts the given value at the beginning of the
                student data file  ( This place in the student data
                file is used to store the number of records the file
                contains )
  ---------------------------------
*/

int Write_num_records( int number )
{
  FILE *random,

  if ( number > 1 ) {
    /*
      Open disk file to write number of student records
    */
    if (( random = fopen ( FILENAME, "r+b" )) != NULL) {
      /*
        Position file pointer at beginning of file
      */
      fseek( random, 0L, SEEK_SET ),
      /*
        Write integer value at beginning of file
      */
      putw ( number, random ),

      fclose( random ),             /* Close disk file        */
      return( 1 ),                  /* Write successfull      */
      }
    else
      return( 0 ),                  /* Write unsuccessfull    */
  }
  else {
```

```
    /*
      Create student data file
    */
    if (( random = fopen ( FILENAME, "wb" )) != NULL) {
      /*
        Write integer value at beginning of file
      */
      putw ( number, random ),

      fclose( random ),              /* Close disk file          */
      return( 1 ),                   /* Write successfull        */
      }
    else
      return( 0 ),                   /* Write unsuccessfull      */
  }
}


/*
  Function to return number of
  records in disk file
*/

int Num_records( void )
{
  FILE *random,
  int nrecs,

  /* Open disk file to read number of student records */
  if (( random = fopen ( FILENAME, "rb" )) != NULL) {
    nrecs = getw ( random );          /* Get number of records     */
    fclose( random ),                 /* Close disk file           */
    return( nrecs ),                  /* Return number of records  */
    }
  else
    return( 0 ),                   /* File open failed! return 0 */
}


/*
  Function to read information in
  student index file into linked
  list
*/

int Index_to_link_list( int recs, NODE **h, NODE **t )
{
  FILE *check,
  INDEX_INFO record,
  int result,
  int counter,

  /* Open index file on disk */
  if (( check = fopen( INDEX, "rb" )) != NULL) {

    /* loop size defined by number of recs to read */
    for ( counter = 1, counter <= recs, counter++ ) {

      /* read index record from disk */
      fread( &record, sizeof( INDEX_INFO ), 1, check ),

      /* insert index record into linked list */
      addsl( record offset, h, t, record qualifier ),
    }
    fclose( check ),                  /* Close disk file          */
    return( 1 ),                   /* Read successful return 1  */
  }
  else
    return( 0 );                   /* File open failed! return 0 */
```

```
}


/*
  Function to read student record
  from student data file on disk
*/

void Fetch( long st_offset, STUDENT_RECORD *buffer )
{
  FILE *random,
  int result,
  int *save_error_box,

  /* Open student data disk file */
  if (( random = fopen ( FILENAME, "rb" )) != NULL ) {

    /* Set file offset pointer in disk file to st_offset */
    result = fseek( random, st_offset, SEEK_SET ),

    /* Determine whether seek was successful */
    if ( result != 0 ) {
      /* Seek failed! */

      /*
        set error box color to red
        set error text color to white
      */
      menu_back_color( BK_RED );
      menu_text_color( T_WHITE | T_BRIGHT ),

      /* Display error_box_1_03 */
      save_error_box = menu_message( 10, 8, error_box_1_03 ),

      /* Error Sound */
      warble( 5 );

      /* Get key/mouse press from user */
      getkey_or_mouse(),

      /* Erase error_box_1 */
      menu_erase( save_error_box );

      /*
        set box color back to cyan
        set text color back to black
      */
      menu_back_color( BK_WHITE ),
      menu_text_color( T_BLACK ),
      }
    else {
      /* Seek successful */
      /* Read student data record into buffer */
      fread( buffer, sizeof( STUDENT_RECORD ), 1  random ),
      }

    /* Close disk file */
    fclose( random ),
  }
}


/* ---------------------------------
  Function      Save_student_record(),
  File          FILE C

  Parameters
   (input)     flag            0 = student has record on disk
```

```
                        1 = student does not have record
                          on disk

            buffer        student record to save

  Returned      (function returns nothing)

  Variables     random          logical name for student data file
                nrecs           number of records in student data
                                file
                counter         loop counter
                result          error flag

  Description   Saves student record to the student data file  Handles
                the two conditions of the student having a record
                on disk, and the student not having a record on disk
  ---------------------------------
*/

void Save_student_record( long offset, STUDENT_RECORD *buffer )
{
  FILE *random,
  FILE *tmp,
  int nrecs,
  int counter,
  int result,
  int *save_error_box,

  /*
    Save new student record
  */
  if ( offset == 0L ) {
    /*
      Get number of records
    */
    nrecs = Num_records(),

    /*
      Update header in student data file that contains
      the number of student records the file contains
    */
    ++nrecs,
    result = Write_num_records( nrecs ),

    /*
      Open disk file to append student record
    */
    if (( random = fopen ( FILENAME, "ab" )) != NULL) {

      /*
        Append student record
      */
      fwrite( buffer, sizeof( STUDENT_RECORD ), 1, random ),

      /* close file */
      fclose ( random ),
    }
    else {
      /*
        Handle possible errors
      */
      /*
        set error box color to red
        set error text color to white
      */
      menu_back_color( BK_RED ),
      menu_text_color( T_WHITE | T_BRIGHT ),

      /* Display error_box_1_05 */
```

```
  {
                    case 0   image = aircraft_ptr[0], break,

                    case 45  image = aircraft_ptr[1], break,

                    case 90  image = aircraft_ptr[2], break,

                    case 135 image = aircraft_ptr[3], break,

                    case 180 image = aircraft_ptr[4], break,

                    case 225 image = aircraft_ptr[5], break,

                    case 270 image = aircraft_ptr[6], break,

                    case 315 image = aircraft_ptr[7], break,
  }

  /* determine aircraft position required relative to center of screen
          North (0 deg bearing) being up on the screen
  */
  switch( ac_position )
  {
                    case 0    x = 400, y = 550, break,

                    case 45   x = 575, y = 475, break,

                    case 90   x = 650, y = 300, break,

                    case 135  x = 575, y = 125, break,

                    case 180  x = 400, y = 50,  break,

                    case 225  x = 225, y = 125, break,

                    case 270  x = 150, y = 300, break,

                    case 315  x = 225, y = 475, break,

  }

  /* place aircraft image on screen */
  _putimage( device_x( x-25 ), device_y( y+25 ), image, _GPSET ),

}
/*--------------------------------
  Function     Draw_example_aircraft_problem()

  File         t1object c

  Parameters   orientation of aircraft
                                        position of aircraft on screen

  Returned     None

  Description  draws the aircraft on screen at the position and
                                        and orientation specified

  --------------------------------
*/
void Draw_example_aircraft_problem( short ac_orientation, short ac_position )
{

  char *image,
  short x, y,

  /* determine aircraft orientation required */
  switch( ac_orientation )
  {
```

```
        /* Erase error_box_1 */
        menu_erase( save_error_box ),

        /*
          set box color back to cyan
          set text color back to black
        */
        menu_back_color( BK_WHITE ),
        menu_text_color( T_BLACK ),
      }

    }
}


/* ---------------------------------
  Function      Create_index_file
  File          FILE C

  Parameters
   (input)

  Returned      (function returns nothing)

  Variables

  Description   Create index file on disk from student data file
                on disk
  ---------------------------------
*/

void Create_index_file( void )
{
  int  nrecs, rec,
  int *save_error_box,
  FILE  *fil, *ndx,
  INDEX_INFO  ndex,
  STUDENT_RECORD st_rec,

  /*
    Open student data file
  */
  if (( fil = fopen( FILENAME, "rb" )) != NULL ) {

    /*
      Get number of records in file
    */
    nrecs = getw( fil ),

    /*
      Create index file
    */
    ndx = fopen( INDEX, "wb" ),

    for ( rec = 1, rec <= nrecs  rec++ ) {

      /*
        read file position
      */
      ndex offset = ftell( fil )

      /*
        retrieve record from student data file
      */
      fread( &st_rec, sizeof( st_rec ), 1, fil ),

      /*
        copy student record qualifier to index qualifier
      */
```

```
    strcpy( ndex qualifier, st_rec qualifier ),

    /*
      write index record to index file
    */
    fwrite( &ndex, sizeof( ndex ), 1, ndx ),
  }

/*
  close opened files
*/
fclose( ndx ),
fclose( fil ),

}
else {

  /*
    close opened file
  */
  fclose( fil ),

  if ( Num_records() != 0 ) {

    /*
      Handle possible errors
    */
    /*
      set error box color to red
      set error text color to white
    */
      menu_back_color( BK_RED ),
      menu_text_color( T_WHITE | T_BRIGHT ),

    /* Display error_box_1_07 */
    save_error_box = menu_message( 10, 8, error_box_1_07 ),

    /* Error Sound */
    warble( 5 ),

    /* Get key/mouse press from user */
    getkey_or_mouse(),

    /* Erase error_box_1_07 */
    menu_erase( save_error_box ),

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE ),
    menu_text_color( T_BLACK ),
```

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include "typ_init.h"

main()
{
        FILE        *file_handle,
                                                *output_file1,
                                                *output_file2,
                                                *output_file3;
        STUDENT_RECORD data;

        int   number_of_files, count, count1, count2;
        char  info[11];

        /* display general info */
        system("cls");
        printf("Mental Rotation Test Filesee 1.10 Written By Animesh Banerjee\n");
        printf("Adapted from ATC Filesee 2.1 Written By Gordon Jones\n\n");
        printf("23 May, 1996\n");
        printf("<Press any key to run program>");
        getch();

        system("cls");

        /* run the main program */
        printf("Mental Rotation Test Filesee 1.0 Written By Animesh Banerjee\n\n");
        printf("Student Data Convertion to MS Excel 3.0 for Windows 3.X Started...\n\n");
        if ( (output_file1 = fopen("std_1_1.xls", "wt") ) == NULL )

                exit(-1);

        if ( (output_file2 = fopen("std_1_2.xls", "wt") ) == NULL )

                exit(-1);

        if ( (output_file3 = fopen("std_1_3.xls", "wt") ) == NULL )

                exit(-1);

        if ( (file_handle = fopen("student.fil", "rb") ) == NULL )

                exit(-1);

        else
        {
                number_of_files = getw( file_handle );

                fprintf(output_file1, "Student Test Info:- Correct Answer Times\n\n");
                fprintf(output_file1, "Subject #,Sex,Test 1 #1, #2\n\n");
                fprintf(output_file2, "Student Test Info:- Incorrect Answer Times\n\n");
                fprintf(output_file2, "Subject #,Sex,Test 1 #1, #2\n\n");
                printf("\nWriting Student Data on Test #1 - Mental Rotation\n");
                fprintf(output_file3, "Mental Rotation/Orientation Test:- Raw Data\n\n");

                fprintf(output_file3,"                  ");
                for ( count1 = 1; count1 <= 64; count1++ )
                {
                        fprintf( output_file3, "Problem #%2d        " ,count1);
                }
                fprintf(output_file3, "\n");
                fprintf( output_file3, "\nSubject #   Sex   ");
                for ( count1 = 1; count1 <= 64; count1++ )
                {
                        fprintf( output_file3, "reac_time  r/w ans  " );
                }
```

```
                for ( count = 1; count <= number_of_files; count++ )
                {
                        fread( &data, sizeof( STUDENT_RECORD ), 1, file_handle );
                        printf("\tWriting Data for Student #%s\n", data.qualifier);
                        fprintf( output_file1, "%s%c,", data.qualifier, data.male_female );
                        fprintf( output_file2, "%s%c,", data.qualifier, data.male_female );
                        fprintf( output_file3, "\n%s    %c   ", data.qualifier, data.male_female );


                        for ( count1 = 0; count1 < 2; count1++ )
                        {
                                /* change to 1 because only 2 trials */
                                sprintf( info, "%f", data.student_info[count1].avg_time_correct );
                                fprintf( output_file1, "\t%s", info );
                                sprintf( info, "%f", data.student_info[count1].avg_time_incorrect );
                                fprintf( output_file2, "\t%s", info );
                        }
                        fprintf( output_file1, "\n" );
                        fprintf( output_file2, "\n" );

                        for ( count2 = 0; count2 <= 10; count2++ )
                                info[count2] = '\0';

                        for ( count1 = 0; count1 < 64; count1++ )
                        {
                                sprintf( info, "%7.2f     ", data.RESPONSE[count1].reaction_time);
                                fprintf( output_file3, "%s", info );
                                sprintf( info, "%d   ", data.RESPONSE[count1].right_wrong);
                                fprintf( output_file3, "%s", info );
                                sprintf( info, "%c  ", data.RESPONSE[count1].answer );
                                fprintf( output_file3, "%s", info );
                        }
                }
                fprintf( output_file3, "\n" );

        }

        fclose( output_file1 ); fclose( file_handle );fclose( output_file2 );
        fclose( output_file3 );

        return 0;
}
```

```
        Description:    Returns an unsigned integer that corresponds to a
                                                keypress; also detects mouse motion and converts
                                                it to equivalent keypresses
        --------------------------------
*/

unsigned getkey_or_mouse( void )
{
        unsigned key;
        int status, buttons;
        int horz, vert;
        int presses, horz_pos, vert_pos;
        int tot_horz, tot_vert;

        /* Set the mouse motion counters to 0 */
        tot_horz = tot_vert = 0;

        /* Clear out the mouse button press counts */
        mouse_press( LBUTTON, &status, &presses, &horz_pos, &vert_pos );
        mouse_press( RBUTTON, &status, &presses, &horz_pos, &vert_pos );

        /* Loop starts here, watches for keypress or mouse activity */
        while ( 1 )
                {

                switch ( mouse_flag )
                        {

                        /* If this is first iteration, check for existence of mouse */
                        case 0:
                                mouse_reset( &status, &buttons );
                                if ( status == 0 )
                                        mouse_flag = -1;
                                else
                                        mouse_flag = 1;
                                break;

                        /* If mouse does not exist, ignore monitoring functions */
                        case -1:
                                break;

                        /* Check for mouse activity */
                        case 1:

                                /* Accumulate mouse motion counts */
                                mouse_motion( &horz, &vert );
                                tot_horz += horz;
                                tot_vert += vert;

                                /* Check for enough horizontal motion */
                                if ( tot_horz < -HORZ_COUNTS )
                                        return ( KEY_LEFT );
                                if ( tot_horz > HORZ_COUNTS )
                                        return ( KEY_RIGHT );

                                /* Check for enough vertical motion */
                                if ( tot_vert < -VERT_COUNTS )
                                        return ( KEY_UP );
                                if ( tot_vert > VERT_COUNTS )
                                        return ( KEY_DOWN );

                                /* Check for mouse left button presses */
                                mouse_press( LBUTTON, &status, &presses,

        &horz_pos, &vert_pos );
                                if ( presses )
                                        return ( KEY_ENTER );

                                /* Check for mouse right button presses */
```

```
*/

long student_timer( int *key, char *neutral, unsigned timeout, unsigned warning_time )
{
        int flag = 0,
        int beep_flag = 1,
        char *temp,
        clock_t cstart, cend, ct_time,
        enum boolean { TIMEOUT_ENABLED = 1, TIMEOUT_DISABLED = 0},
        enum boolean status,
        long starttime,
        long current_time,
        long elapsed_time = 0,
        long endtime,
        long timetaken,

        cstart = clock(), starttime = cstart,

        /* determine if timeout feature is enabled */
        if ( timeout > 0 )
                status = TIMEOUT_ENABLED,
        else
                status = TIMEOUT_DISABLED,

        while(1)
        {
                if ( status == TIMEOUT_ENABLED )
                {
                        ct_time = clock(), current_time = ct_time,

                        /* calculate elapsed time and correct for system clock reset */
                        if ( current_time < starttime )
                        {
                                elapsed_time = 65535 0 - starttime,
                                elapsed_time = elapsed_time + current_time,
                        }
                        else
                                elapsed_time = current_time - starttime,

                        /* check that warning 'beep' feature is enabled */
                        if ( warning_time > 0 )
                        {
                                /* check if warning should be issued */
                                if ( ((long)(timeout - warning_time) <=
                                                                elapsed_time/CLK_TCK) && beep_flag)
                                {
                                        note(2000,2),
                                        /* set beep_flag so only one beep is issued to signal warning */
                                        beep_flag = 0,
                                }
                        }

                        /* check for timeout expiry */
                        if ( elapsed_time/CLK_TCK >= (long)timeout )
                        {
                                *key = 0,
                                return( elapsed_time ),

                        }

                }

                if ( kbhit() )
                {
                        cend = clock(), endtime = cend
                        flag = 0,
                        temp = neutral,
                        *key = getch(),
                        while  ( *temp != '*' && flag < 1)
```

```
                {
                        if ( *key == *temp )
                                ++flag;
                        ++temp;
                }
                if ( flag > 0 )
                {
                        if ( endtime < starttime )
                        {
                                timetaken = 65535.0 - starttime;
                                timetaken = timetaken + endtime;
                        }
                        else
                                timetaken = endtime - starttime;

                        /* check that reaction time is not too low */
                        if ( timetaken >= MIN_REACTION_TIME/1000.0 * CLK_TCK )
                                return( timetaken );
                }
        }
    }
}
```

```
/* ---------------------------------------------------
        Name        LIST C
  Type      Linked list manipulation module for
        Air Traffic Control Screening Program
        Language    Microsoft QuickC version 2

        Last Revision  06/16/92  Gordon Jones
  ---------------------------------------------------
*/

#include <malloc h>              /* Memory allocation routines */
#include <stdio h>               /* Standard input/output      */
#include <string h>              /* String manipulation        */
#include "typ_init h"            /* structure definitions for  */
                                 /* STUDENT_COLUMN             */
#include "list h"                /* Linked list routines       */


/* ---------------------------------
        Function    Addsl
        File        LIST C

        Parameters
                (input)   offset    offset in bytes where student record
                                                                          is in file
<student fil>
                (input)   h         pointer to head of linked list
                (input)   t         pointer to tail of linked list
                (input)   key       student identifier to be added to
                                                                          linked
list

        Returned    None

        Variables   new       pointer to tempory record

        Description  Procedure to add a record (node) to tail of linked
                                                          list

        Note        For additional help refer to any data structures
                                                          book on singly linked lists -> simplest!
        ---------------------------------
*/

void addsl( long offset, NODE **h, NODE**t, char *key )
{
  NODE *new,

  new = malloc( sizeof ( NODE )),
  new->offset = offset,            /* copy offset into node offset   */
  strcpy( new->qualifier, key ),   /* copy qualifier into node       */
  if( *t != NULL )
    ( *t ) -> next = new,          /* update old tail's pointer field */
  if( *h == NULL )
    ( *h ) = new,                  /* set head pointer if necessary  */
  *t = new,                        /* update tail pointer            */
  ( *t ) -> next = NULL,           /* blank new tail's pointer field */
}


/* ---------------------------------
        Function    Freelist
        File        LIST C

        Parameters
                (input)   h         pointer to head of linked list

        Returned    None
```

```
            Variables     n          pointer to tempory record

            Description   Procedure to delete linked list from memory

            Note          For additional help refer to any data structures
                                                    book on singly linked lists -> simplest!
            ---------------------------------
*/

void freelist( NODE *h )
{
  NODE *n,

  n = h,                    /* point to head of list          */
  while( n != NULL ) {          /* loop until end of list          */
    free(n),                /* free current node              */
    n = n->next,              /* go to next node                */
  }
}


/* ---------------------------------
            Function      Check
            File          LIST C

            Parameters
                  (input)   h         pointer to head of linked list
                  (input)   key       pointer to field containing
                                                                                    student
identifier

            Returned      offset in bytes of student record in student file
                                                    <student fil>
                                                    if record not found 0 is returned

            Variables     n          pointer to tempory record

            Description   Procedure to determine if student record with
                                                    student identifier <key> is in linked list  If
                                                    it is return value of offset field (offset of
                                                    student record in student file <student fil> in
                                                    bytes

            ---------------------------------
*/

long check( NODE *h, char *key )
{
  NODE *n,

  n = h,                    /* point to head of list          */
  while( n != NULL ) {          /* loop until end of list          */
    if ( strcmp ( n->qualifier, key ) == 0 )
      return( n->offset ),      /* qualifier = key then return     */
    n = n->next,              /* offset in disk file            */
  }
  return( 0L ),               /* qualifier not found => return 0  */
}


/* ---------------------------------
            Function      Res_addsl
            File          LIST C

            Parameters
                  (input)   n         pointer to student record to be added
                                                                                    to linked
list
                  (input)   h         pointer to head of linked list
```

```
                (input)   t          pointer to tail of linked list

        Returned      None

        Variables     new        pointer to tempory record
                                                    counter     tempory loop counter

        Description   Procedure to add a record (node) to tail of linked
                                                    list

        Note          For additional help refer to any data structures
                                                    book on singly linked lists -> simplest!
        ---------------------------------
*/

void res_addsl( STUDENT_RECORD *n, RES_NODE **h, RES_NODE **t )
{
  RES_NODE *new,
           register int counter,

  new = malloc( sizeof ( RES_NODE )),

  /*
    copy record passed to procedure ( STUDENT_RECORD *n )
    into node of type RES_NODE and then add this
    node to linked list
  */
           for( counter = 0, counter <= 29, counter++ )
    new->student_info[counter] = n->student_info[counter],
  strcpy( new->qualifier, n->qualifier ),
  new->r_l_handed = n->r_l_handed,
  new->test_no = n->test_no,

  if( *t != NULL )
    ( *t ) -> next = new,          /* update old tail's pointer field */
  if( *h == NULL )
    ( *h ) = new,                  /* set head pointer if necessary   */
  *t = new,                        /* update tail pointer             */
  ( *t ) -> next = NULL,           /* blank new tail's pointer field  */
}


/* ---------------------------------
        Function      Res_freelist
        File          LIST C

        Parameters
                (input)   h          pointer to head of linked list

        Returned      None

        Variables     n          pointer to tempory record

        Description   Procedure to delete linked list from memory

        Note          For additional help refer to any data structures
                                                    book on singly linked lists -> simplest!
        ---------------------------------
*/

void res_freelist( RES_NODE *h )
{
  RES_NODE *n,

  n = h,                       /* point to head of list          */
  while( n != NULL ) {             /* loop until end of list         */
    free(n),                   /* free current node              */
    n = n->next,                 /* go to next node                */
  }
```

```
/* ---------------------------------------------
   Name          MENU C
   Type          Toolbox module
   Language      Microsoft QuickC
                 Video          (no special video requirements)
   ---------------------------------------------
*/

#include <graph h>
#include <stdio h>
#include <ctype h>
#include <string h>
#include <malloc h>
#include "box h"
#include "mousefun h"
#include "getkey h"
#include "t_colors h"
#include "menu h"

/* Default menu colors */
static int c_lines = T_BLACK,
static int c_title = T_BLACK,
static int c_text = T_BLACK,
static int c_prompt = T_BLACK,
static int c_hitext = T_WHITE,
static int c_hiletter = T_WHITE | T_BRIGHT,
static long int c_back = BK_WHITE,
static long int c_hiback = BK_BLACK,

/* Default border lines and shadow control */
static int mb_lines = 1,
static int mb_shadow = 1,


/* ---------------------------------
   Function       menu_box_lines()

   Parameters
    (input)       line_type          0, 1, or 2 (outline)

   Returned       (function returns nothing)

   Variables      (none)

   Description    Sets the box outline type  Selects single-line or
                  double-line border (or none)
   ---------------------------------
*/

void menu_box_lines( int line_type )
{
   mb_lines = line_type,
}


/* ---------------------------------
   Function       menu_box_shadow()

   Parameters
    (input)       on_off             Shadow control

   Returned       (function returns nothing)

   Variables      (none)

   Description    Sets the menu box shadow control to on or off
                  0 = off, non-zero = on
   ---------------------------------
*/
```

```
void menu_box_shadow( int on_off )
{
   mb_shadow = on_off,
}


/* ---------------------------------
   Function       menu_back_color()

   Parameters
    (input)     back            Background color

   Returned       (function returns nothing)

   Variables      (none)

   Description    Sets the background color for boxes
   ---------------------------------
*/

void menu_back_color( long back )
{
   c_back = back,
}


/* ---------------------------------
   Function       menu_line_color()

   Parameters
    (input)     lines           Border line color

   Returned       (function returns nothing)

   Variables      (none)

   Description    Sets the box outline color
   ---------------------------------
*/

void menu_line_color( int lines )
{
   c_lines = lines,
}


/* ---------------------------------
   Function       menu_title_color()

   Parameters
    (input)     title           Title text color

   Returned       (function returns nothing)

   Variables      (none)

   Description    Sets the text color for the title
   ---------------------------------
*/

void menu_title_color( int title )
{
   c_title = title,
}


/* ---------------------------------
   Function       menu_text_color()
```

```
  Parameters
    (input)     text            Menu text color

  Returned      (function returns nothing)

  Variables     (none)

  Description   Sets the menu box text color
  ---------------------------------
*/

void menu_text_color( int text )
{
   c_text = text,
}


/* ---------------------------------
  Function      menu_prompt_color()

  Parameters
    (input)     prompt          Menu prompt line color

  Returned      (function returns nothing)

  Variables     (none)

  Description   Sets the menu box prompt line text color
  ---------------------------------
*/

void menu_prompt_color( int prompt )
{
   c_prompt = prompt,
}


/* ---------------------------------
  Function      menu_hilight_letter()

  Parameters
    (input)     hiletter        Highlighted letter color

  Returned      (function returns nothing)

  Variables     (none)

  Description   Sets highlighted character color for menu options
  ---------------------------------
*/

void menu_hilight_letter( int hiletter )
{
   c_hiletter = hiletter,
}


/* ---------------------------------
  Function      menu_hilight_text()

  Parameters
    (input)     hitext          Highlighted text color

  Returned      (function returns nothing)

  Variables     (none)

  Description   Sets highlighted text color for menu options
```

```
  ---------------------------------
*/

void menu_hilight_text( int hitext )
{
   c_hitext = hitext,
}


/* ---------------------------------
   Function       menu_hilight_back()

   Parameters
    (input)      hiback          Highlighted line background

   Returned      (function returns nothing)

   Variables     (none)

   Description   Sets the background color for the highlighted line
                 in the menu box
   ---------------------------------
*/

void menu_hilight_back( long hiback )
{
   c_hiback = hiback,
}


/* ---------------------------------
   Function       menu_bar()

   Parameters
    (input)      row      Screen row to locate menu bar
    (input)      col      Screen column to locate menu bar
    (input)      string   String of menu bar selections
    (output)     choice   Number of item selected by user

   Returned      Buffer used to restore the background

   Variables     len          Length of menu string
                 fore         Saves current foreground color
                 maxchoice    Number of choices
                 i            Looping index
                 j            Looping index
                 cpos         Current position in the menu
                 quit_flag    Signals to exit function
                 savebuf      Buffer containing background
                 fstr         Foreground color attributes
                 lastc        Last character checked
                 thisc        Current character checked
                 bstr         Background color attributes
                 key          Key code from getkey_or_mouse()
                 back         Saves current background color
                 oldpos       Saves the cursor position

   Description   Creates a pop-up menu bar
   ---------------------------------
*/

int far *menu_bar ( int row  int col, char *string, int *choice )
{
   int len,
   int fore,
   int maxchoice,
   int i, j,
   int cpos,
   int quit_flag = 0,
```

```
int far *savebuf;
int fstr[81];
char lastc, thisc;
long int bstr[81];
unsigned key;
long int back;
struct rccoord oldpos;

/* Save the current color settings */
fore = _gettextcolor();
back = _getbkcolor();

/* Save the current cursor position */
oldpos = _gettextposition();

/* Calculate the string length only once */
len = strlen( string );

/* Save the menu background */
if ( mb_shadow )
                    savebuf = box_get( row, col, row + 1, col + len + 1 );
else
                    savebuf = box_get( row, col, row, col + len - 1 );

/* Put the menu bar on the screen */
_settextposition( row, col );
_outtext( string );

/* Cast a shadow */
if ( mb_shadow )
   {
   _settextcolor( T_GRAY );
   _setbkcolor( BK_BLACK );
   box_color( row + 1, col + 2, row + 1, col + len + 1 );
   }

/* Initialize choice if necessary */
if ( *choice < 1 )
   *choice = 1;

/* Process each key press */
while ( !quit_flag )
   {

   /* Determine the color attributes */
   j = 0;
   maxchoice = 0;
   lastc = 0;
   for ( i = 0; i < len; i++ )
      {
      thisc = string[i];
      if ( lastc == ' ' && thisc == ' ' && i < len - 1 )
         {
         j++;
         maxchoice++;
         }
      if ( j == *choice && i < len - 1 )
         {
         fstr[i] = c_hitext;
         bstr[i] = c_hiback;
         }
      else
         {
         fstr[i] = c_text;
         bstr[i] = c_back;
         }
      if ( isupper( thisc ))
         {
         fstr[i] = c_hiletter;
```

```
      if ( j == *choice )
         cpos = i;
      }
   lastc = thisc;
   }

/* Put the attributes to video */
for ( i = 0; i < len; i++ )
   {
   _settextcolor( fstr[i] );
   _setbkcolor( bstr[i] );
   box_color( row, col + i, row, col + i );
   }

/* Put cursor at appropriate position */
_settextposition( row, col + cpos );

key = getkey_or_mouse();

/* Convert to upper case */
if ( key >= 'a' && key <= 'z' )
   key -= 32;

/* Check for alpha key */
if ( key >= 'A' && key <= 'Z' )
   {
   for ( i = 0; i < len; i++ )
      {
      if ( ++cpos >= len )
         {
         cpos = 0;
         *choice = 0;
         }
      if ( isupper( string[cpos] ))
         *choice += 1;
      if ( string[cpos] == (char)key )
         break;
      }
   }

/* Check for control keys */
switch( key )
   {
   case KEY_LEFT:
      if ( *choice > 1 )
         *choice -= 1;
      break;
   case KEY_RIGHT:
      if ( *choice < maxchoice )
         *choice += 1;
      break;
   case KEY_HOME:
      *choice = 1;
      break;
   case KEY_END:
      *choice = maxchoice;
      break;
   case KEY_ESCAPE:
   case KEY_UP:
      *choice = 0;
      quit_flag = 1;
      break;
   case KEY_ENTER:
   case KEY_DOWN:
      quit_flag = 1;
      break;
   }
}
```

```
   /* Restore original conditions */
   _settextposition( oldpos.row, oldpos.col ),
   _settextcolor( fore ),
   _setbkcolor( back ),
   return ( savebuf ),
}


/* ---------------------------------
   Function       menu_drop()

   Parameters
     (input)      row      Screen row to locate menu bar
     (input)      col      Screen column to locate menu bar
     (input)      strary   String array of menu selections
     (output)     choice   Number of item selected by user

   Returned       Buffer used to restore the background

   Variables      n            Number of strings in menu
        len          Length of menu string
              fore         Saves current foreground color
              tmpcol       Column to start title and prompt
              maxchoice    Number of choices
              i            Looping index
              quit_flag    Signals to exit function
              savebuf      Buffer containing background
              key          Key code from getkey_or_mouse()
              back         Saves current background color
              oldpos       Saves the cursor position

   Description    Creates a popup drop down menu
   ---------------------------------
*/

int far *menu_drop( int row, int col, char **strary, int *choice )
{
   int n = 0,
   int len = 0,
   int fore,
   int tmpcol,
   int maxchoice,
   int i,
   int quit_flag = 0,
   int far *savebuf,
   unsigned key,
   long int back,
   struct rccoord oldpos,

   /* Save the current color settings */
   fore = _gettextcolor(),
   back = _getbkcolor(),

   /* Save the current cursor position */
   oldpos = _gettextposition(),

   /* Determine the number of strings in the menu */
   while ( strary[n] != NULL )
      n++,

   /* Set the maximum choice number */
   maxchoice = n - 2,

   /* Determine the maximum menu string length */
   for ( i = 0, i < n, i++ )
      if ( strlen( strary[i] ) > len )
         len = strlen( strary[i] ),

   /* Save the menu background */
```

```
if ( mb_shadow )
                    savebuf = box_get( row, col, row + n, col + len + 5 ),
else
                    savebuf = box_get( row, col, row + n - 1, col + len + 3 ),

/* Create the menu box */
_settextcolor( c_lines ),
_setbkcolor( c_back ),
box_erase( row, col, row + n - 1, col + len + 3 ),
box_draw( row, col, row + n - 1, col + len + 3, mb_lines ),

/* Cast a shadow */
if ( mb_shadow )
   {
   _settextcolor( T_GRAY ),
   _setbkcolor( BK_BLACK ),
   box_color( row + n, col + 2, row + n, col + len + 3 ),
   box_color( row + 1, col + len + 4, row + n, col + len + 5 ),
   }

/* Put the title at the top */
tmpcol = col + ( len - strlen( strary[0] ) + 4 ) / 2,
_settextposition( row, tmpcol ),
_settextcolor( c_title ),
_setbkcolor( c_back ),
_outtext( strary[0] ),

/* Print the choices */
_settextcolor( c_text ),
for ( i = 1, i <= maxchoice, i++ )
   {
   _settextposition( row + i, col + 2 ),
   _outtext( strary[i] ),
   }

/* Put the prompt at the bottom */
tmpcol = col + ( len - strlen( strary[n - 1] ) + 4 ) / 2,
_settextposition( row + n - 1, tmpcol ),
_settextcolor( c_prompt ),
_outtext( strary[n - 1] ),

/* Initialize choice */
*choice = 1,

/* Process each key press */
while ( !quit_flag )
   {

   /* Determine and set the color attributes */
   for ( i = 1, i <= maxchoice, i++ )
      {
      if ( i == *choice )
         {
         _setbkcolor( c_hiback )
         _settextcolor( c_hiletter ),
         box_color( row + i, col + 1, row + i, col + 2 ),
         _settextcolor( c_hitext ),
         box_color( row + i, col + 3, row + i, col + len + 2 )
         }
      else
         {
         _setbkcolor( c_back ),
         _settextcolor( c_hiletter ),
         box_color( row + i, col + 1, row + i, col + 2 ),
         _settextcolor( c_text ),
         box_color( row + i, col + 3, row + i, col + len + 2 ),
         }
      }
```

```
        /* Put cursor at appropriate position */
        _settextposition( row + *choice, col + 2 ),

        key = getkey_or_mouse(),

        /* Convert to upper case */
        if ( key >= 'a' && key <= 'z' )
           key -= 32,

        /* Check for alpha key */
        if ( key >= 'A' && key <= 'Z' )
           {
           for ( i = 1, i <= maxchoice, i++ )
              {
              *choice += 1,
              if ( *choice > maxchoice )
                 *choice = 1,
              if ( strary[*choice][0] == (char)key )
                 break,
              }
           }

        /* Check for control keys */
        switch ( key )
           {
           case KEY_UP
              if ( *choice > 1 )
                 *choice -= 1,
              break,
           case KEY_DOWN
              if ( *choice < maxchoice )
                 *choice += 1,
              break,
           case KEY_HOME
              *choice = 1,
              break,
           case KEY_END
              *choice = maxchoice,
              break,
           case KEY_ESCAPE
              *choice = 0,
              quit_flag = 1,
              break,
           case KEY_ENTER
              quit_flag = 1,
              break,
           }
        }

     /* Restore original conditions */
     _settextposition( oldpos.row, oldpos.col ),
     _settextcolor( fore ),
     _setbkcolor( back ),
     return ( savebuf ),
}


/* ---------------------------------
   Function      menu_message()

   Parameters
     (input)     row      Screen row to locate message box
     (input)     col      Screen column to locate message box
     (input)     strary   String array of message text

   Returned      Buffer used to restore the background

   Variables     n        Number of strings in message
       len       Length of longest menu string
```

```
            fore        Saves current foreground color
            tmpcol      Column to start title and prompt
            i           Looping index
            savebuf     Buffer containing background
            key         Key code from getkey_or_mouse()
            back        Saves current background color
            oldpos      Saves the cursor position

   Description   Creates a pop-up message box
   ------------------------------------
*/

int far *menu_message( int row, int col, char **strary )
{
   int n = 0,
   int len = 0,
   int fore;
   int tmpcol,
   int i,
   int far *savebuf,
   unsigned key,
   long int back,
   struct rccoord oldpos,

   /* Save the current color settings */
   fore = _gettextcolor(),
   back = _getbkcolor(),

   /* Save the current cursor position */
   oldpos = _gettextposition(),

   /* Determine the number of strings in the message */
   while ( strary[n] != NULL )
      n++,

   /* Determine the maximum message string length */
   for ( i = 0, i < n, i++ )
      if ( strlen( strary[i] ) > len )
         len = strlen( strary[i] ),

   /* Save the message background */
   if ( mb_shadow )
      savebuf = box_get( row, col, row + n, col + len + 5 ),
   else
      savebuf = box_get( row, col, row + n - 1, col + len + 3 ),

   /* Create the information box */
   _settextcolor( c_lines ),
   _setbkcolor( c_back ),
   box_erase( row, col, row + n - 1, col + len + 3 ),
   box_draw( row, col, row + n - 1, col + len + 3, mb_lines ),

   /* Cast a shadow */
   if ( mb_shadow )
      {
      _settextcolor( T_GRAY ),
      _setbkcolor( BK_BLACK ),
      box_color( row + n, col + 2, row + n, col + len + 3 ),
      box_color( row + 1, col + len + 4, row + n, col + len + 5 ),
      }

   /* Put the title at the top */
   tmpcol = col + ( len - strlen( strary[0] ) + 4 ) / 2,
   _settextposition( row, tmpcol ),
   _settextcolor( c_title ),
   _setbkcolor( c_back ),
   _outtext( strary[0] ),

   /* Print the text */
```

```
    _settextcolor( c_text );
    for ( i = 1; i < n - 1; i++ )
       {
       _settextposition( row + i, col + 2 );
       _outtext( strary[i] );
       }

    /* Put the prompt at the bottom */
    tmpcol = col + ( len - strlen( strary[n - 1] ) + 4 ) / 2;
    _settextposition( row + n - 1, tmpcol );
    _settextcolor( c_prompt );
    _outtext( strary[n - 1] );

    /* Restore original conditions */
    _settextposition( oldpos.row, oldpos.col );
    _settextcolor( fore );
    _setbkcolor( back );
    return ( savebuf );
}


/* ---------------------------------
   Function:      menu_erase()

   Parameters:
     (input)      buf      Buffer for restoring background

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Restores the background behind a bar menu,
        pull-down menu, or message box
   ---------------------------------
*/

void menu_erase( int far *buf )
{
    box_put( buf );
    _ffree( buf );
}
```

```
/* ------------------------------------------------------
  Name        MN_MENU C
  Type        Routines that display the main
                menu and the choices that are
                available to the user
                Air Traffic Control Screening Program

  Language    Microsoft QuickC version 2
  ------------------------------------------------------
*/

#include <stdio h>
#include <graph h>
#include <process h>
#include "typ_init h"
#include "file h"
#include "list h"
#include "tmanager h"
#include "menu h"
#include "box h"
#include "t_colors h"
#include "dsk_init h"
#include "data_plt h"
#include "getkey h"


char *error_box_1_08[] =
  {
  " Error Message #1 08 ",
  "",
  " Unable to spawn statistical ",
  " analysis program <st_menu exe>  ",
  "",
  " Result => the program can not ",
  " be loaded and executed  ",
  "",
  " Action => check that st_menu exe ",
  " is located in the same directory ",
    as the other program files ",
  "",
  "< Press any key >",
  NULL
  },

 char *drop_main_menu[] =
  {
  " Main Menu ",
  "Perform Student Tests",
          "Demonstration Tests",
          "Exit Program",
  " Select ",
  NULL
          },

char *drop_sub_menu[] =
          {
          " Practice Menu ",
          "Test #1",
          "Main Menu",
          " ",
          NULL
          },

char *drop_full_menu[] =
          {
          " Perform Student Test Menu ",
          "Test #1",
          "All Tests",
          "  ,
```

```
        NULL
        },



/* --------------------------------
  Function      Display_main_menu
  File          MN_MENU C

  Parameters
          (input)
                  head    pointer to head of student index linked
                                  list of type NODE
                  tail    pointer to tail of student index linked
                                  list of type NODE
                  r_head  pointer to head of student record linked
                                  list of type RES_NODE
                  r_tail  pointer to tail or student record linked
                                  list of type RES_NODE

  Returned      None

  Variables
                  choice    User choice from drop down menu
                  r         Return value from spawn command
                  offset    Offset of student record in student
                                  file held on disk
                  args      arguments passed to the spawn command
                                  args[0] is pointer to filename to be
                                  executed args[1] is NULL pointer to end
                                  of argument list
                  prog      filename to be executed by spawn command

  Description   Displays the main menu and prompts the user to
                  select from one of the choices available
  --------------------------------
*/

STUDENT_RECORD new_student,

void display_main_menu( NODE **head, NODE **tail,
                        RES_NODE **r_head, RES_NODE **r_tail )
{
        int choice = 0,
        int second_choice = 0,
  int r,
  long offset,
  char *args[2],
  char prog[80] = "st_menu",
  int *save_error_box,

  args[0] = prog,
  args[1] = NULL,

        new_student test_no = -10,

        while ( choice != 3 ) {
   /*
        Display main menu
   */
   menu_erase( menu_drop( 4, 18, drop_main_menu, &choice )),

   switch( choice ) {
        case 1
          /* Perform Student Tests */
                                new_student test_no = 0,

          /* Initialize linked list of index to student records on file */
```

```
/* ---------------------------------------------------
   Name          MOUSEFUN C
   Type          Toolbox module
   Language      Microsoft QuickC version 2
   Demonstrated  MOUSTEST C
   Video         Some functions require CGA or better graphics
   ---------------------------------------------------
*/

#include <dos.h>
#include "mousefun.h"

/* ---------------------------------
   Function      mouse_reset()
   Toolbox       MOUSEFUN C
   Demonstrated  MOUSTEST C

   Parameters
    (output)     status   Status of the mouse
    (output)     buttons  Number of mouse buttons

   Returned      (function returns nothing)

   Variables     m1       Local variable for register ax
                 m2       Local variable for register bx

   Description   Resets the mouse and verifies its existence
   ---------------------------------
*/

void mouse_reset( int *status, int *buttons )
{
   int m1, m2,

          _asm
                    {
                    xor  ax, ax
     int  33h
     mov  m1, ax
     mov  m2, bx
     }

   *status = m1,
   *buttons = m2,
}


/* ---------------------------------
   Function      mouse_show()
   Toolbox       MOUSEFUN C
   Demonstrated  MOUSTEST C

   Parameters    (none)

   Returned      (function returns nothing)

   Variables     (none)

   Description   Makes the mouse cursor visible
   ---------------------------------
*/

void mouse_show( void )
{
   _asm
     {
     mov  ax, 1
     int  33h
     }
```

```
}

/*---------------------------------
  Function      mouse_hide()
  Toolbox       MOUSEFUN C
  Demonstrated  MOUSTEST C

  Parameters    (none)

  Returned      (function returns nothing)

  Variables     (none)

  Description   Makes the mouse cursor invisible
  ---------------------------------
*/

void mouse_hide( void )
{
   _asm
      {
      mov  ax, 2
      int  33h
      }
}


/*---------------------------------
  Function      mouse_status()
  Toolbox       MOUSEFUN C
  Demonstrated  MOUSTEST C

  Parameters
   (output)    left_button   State of the left button
   (output)    right_button  State of the right button
   (output)    horz_pos      Horizontal position of the mouse
   (output)    vert_pos      Vertical position of the mouse

  Returned      (function returns nothing)

  Variables     m2      Local variable for register bx
                m3      Local variable for register cx
                m4      Local variable for register dx

  Description   Gets the current state of the mouse buttons and
                the mouse cursor position
  ---------------------------------
*/

void mouse_status( int *left_button, int *right_button,
      int *horz_pos  int *vert_pos )
{
   int m2, m3, m4,

   _asm
      {
      mov  ax, 3
      int  33h
      mov  m2, bx
      mov  m3, cx
      mov  m4, dx
      }

   *left_button = m2 & 1,
   *right_button = ( m2 >> 1 ) & 1,
   *horz_pos = m3,
   *vert_pos = m4,
}
```

```
/* ---------------------------------
   Function      mouse_setpos()
   Toolbox       MOUSEFUN C
   Demonstrated  MOUSTEST C

   Parameters
    (input)     horizontal    Horizontal position
    (input)     vertical      Vertical position

   Returned      (function returns nothing)

   Variables     (none)

   Description   Sets the mouse cursor to the indicated position
   ---------------------------------
*/

void mouse_setpos( int horizontal, int vertical )
{
   _asm
      {
      mov  ax, 4
      mov  cx, horizontal
      mov  dx, vertical
      int  33h
      }
}


/* ---------------------------------
   Function      mouse_press()
   Toolbox       MOUSEFUN C
   Demonstrated  MOUSTEST C

   Parameters
    (input)     button   Left or right button
    (output)    status   Status of the button
    (output)    presses  Number of button presses
    (output)    horz_pos Horizontal position at last press
    (output)    vert_pos Vertical position at last press

   Returned      (function returns nothing)

   Variables     m1       Local variable for register ax
                 m2       Local variable for register bx
                 m3       Local variable for register cx
                 m4       Local variable for register dx

   Description   Gets button press information
   ---------------------------------
*/

void mouse_press( int button, int *status, int *presses,
      int *horz_pos, int *vert_pos )
{
   int m1, m2, m3, m4,

   _asm
      {
      mov  ax, 5
      mov  bx, button
      int  33h
      mov  m1, ax
      mov  m2, bx
      mov  m3, cx
      mov  m4, dx
      }
```

```
    if ( button == LBUTTON )
       *status = m1 & 1,
    else
       *status = ( m1 >> 1 ) & 1,

    *presses = m2;
    *horz_pos = m3,
    *vert_pos = m4,
}


/* ---------------------------------
   Function       mouse_release()
   Toolbox        MOUSEFUN C
   Demonstrated   MOUSTEST C

   Parameters
    (input)      button    Left or right button
    (output)     status    Status of the button
    (output)     presses   Number of button releases
    (output)     horz_pos  Horizontal position at last release
    (output)     vert_pos  Vertical position at last release

   Returned       (function returns nothing)

   Variables      m1        Local variable for register ax
                  m2        Local variable for register bx
                  m3        Local variable for register cx
                  m4        Local variable for register dx

   Description    Gets button release information
   ---------------------------------
*/

void mouse_release ( int button, int *status, int *releases,
                     int *horz_pos, int *vert_pos )
{
    int m1, m2, m3, m4,

    _asm
       {
       mov  ax, 6
       mov  bx, button
       int  33h
       mov  m1, ax
       mov  m2, bx
       mov  m3, cx
       mov  m4, dx
       }

    if ( button == LBUTTON )
       *status = m1 & 1,
    else
       *status = ( m1 >> 1 ) & 1,

    *releases = m2,
    *horz_pos = m3,
    *vert_pos = m4,
}


/* ---------------------------------
   Function       mouse_sethorz()
   Toolbox        MOUSEFUN C
   Demonstrated   MOUSTEST C

   Parameters
    (input)      horz_min  Minimum horizontal cursor position
```

```
    (input)     horz_max  Maximum horizontal cursor position

   Returned     (function returns nothing)

   Variables    (none)

   Description  Sets minimum and maximum horizontal mouse
       cursor positions
   ---------------------------------
*/

void mouse_sethorz( int horz_min, int horz_max )
{
   _asm
      {
      mov  ax, 7
      mov  cx, horz_min
      mov  dx, horz_max
      int  33h
      }
}


/* ---------------------------------
   Function     mouse_setvert()
   Toolbox      MOUSEFUN C
   Demonstrated  MOUSTEST C

   Parameters
    (input)     vert_min  Minimum vertical cursor position
    (input)     vert_max  Maximum vertical cursor position

   Returned     (function returns nothing)

   Variables    (none)

   Description  Sets minimum and maximum vertical mouse cursor
       positions
   ---------------------------------
*/

void mouse_setvert( int vert_min, int vert_max )
{
   _asm
      {
      mov  ax, 8
      mov  cx, vert_min
      mov  dx, vert_max
      int  33h
      }
}


/* ---------------------------------
   Function     mouse_setgcurs()
   Toolbox      MOUSEFUN C
   Demonstrated  MOUSTEST C

   Parameters
    (input)     cursor   Structure defining a graphics cursor

   Returned     (function returns nothing)

   Variables    cursor_seg    Segment of the cursor structure
                cursor_off    Offset of the cursor structure
                hotx          Hot spot x value
                hoty          Hot spot y value

   Description  Creates a graphics mode mouse cursor
```

```
  ---------------------------------
*/

void mouse_setgcurs( struct graphics_cursor far *cursor )
{
   unsigned cursor_seg = FP_SEG( cursor ),
   unsigned cursor_off = FP_OFF( cursor ),
   int hotx = cursor->hot_spot_x,
   int hoty = cursor->hot_spot_y,

   _asm
      {
      mov  ax, 9
      mov  bx, hotx
      mov  cx, hoty
      mov  es, cursor_seg
      mov  dx, cursor_off
      int  33h
      }
}


/* ---------------------------------
  Function      mouse_settcurs()
  Toolbox       MOUSEFUN C
  Demonstrated  MOUSTEST C

  Parameters
   (input)      cursor_select  Hardware or software cursor
   (input)      screen_mask    Screen mask (or start scan line)
   (input)      cursor_mask    Cursor mask (or end scan line)

  Returned      (function returns nothing)

  Variables     (none)

  Description   Sets the text mode hardware or software cursor
  ---------------------------------
*/

void mouse_settcurs( int cursor_select, int screen_mask, int cursor_mask )
{
   _asm
      {
      mov  ax, 10
      mov  bx, cursor_select
      mov  cx, screen_mask
      mov  dx, cursor_mask
      int  33h
      }
}


/* ---------------------------------
  Function      mouse_motion()
  Toolbox       MOUSEFUN C
  Demonstrated  MOUSTEST C

  Parameters
   (output)     horz_mickeys   Horizontal mickeys
   (output)     vert_mickeys   Vertical mickeys

  Returned      (function returns nothing)

  Variables     m3      Local variable for register cx
                m4      Local variable for register dx

  Description   Gets the accumulated mouse motion counts
                (mickeys) since the last call to this function
```

```
  ---------------------------------
*/

void mouse_motion( int *horz_mickeys, int *vert_mickeys )
{
   int m3, m4;

   _asm
      {
      mov   ax, 11
      int   33h
      mov   m3, cx
      mov   m4, dx
      }

   *horz_mickeys = m3;
   *vert_mickeys = m4;
}


/* ---------------------------------
   Function:      mouse_setratios()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
    (output)     horizontal     Horizontal mickey/pixel ratio
    (output)     vertical       Vertical mickey/pixel ratio

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Sets the mickey/pixel ratios for mouse motion
   ---------------------------------
*/

void mouse_setratios( int horizontal, int vertical )
{
   _asm
      {
      mov   ax, 15
      mov   cx, horizontal
      mov   dx, vertical
      int   33h
      }
}


/* ---------------------------------
   Function:      mouse_condoff()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
    (input)      x1       Upper left corner of region
    (input)      y1       Upper left corner of region
    (input)      x2       Lower right corner of region
    (input)      y2       Lower right corner of region

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Sets a region for conditionally turning off the
                  mouse cursor
   ---------------------------------
*/
```

```
void mouse_condoff( int x1, int y1, int x2, int y2 )
{
  _asm
    {
    mov  ax, 16
    mov  cx, x1
    mov  dx, y1
    mov  si, x2
    mov  di, y2
    int  33h
    }
}


/* ---------------------------------
  Function      mouse_setdouble()
  Toolbox       MOUSEFUN C
  Demonstrated  MOUSTEST C

  Parameters
   (input)      mickeys_per_second  Double speed threshold

  Returned      (function returns nothing)

  Variables     (none)

  Description   Sets the mouse double speed threshold
  ---------------------------------
*/

void mouse_setdouble( int mickeys_per_second )
{
  _asm
    {
    mov  ax, 19
    mov  dx, mickeys_per_second
    int  33h
    }
}


/* ---------------------------------
  Function      mouse_storage()
  Toolbox       MOUSEFUN C
  Demonstrated  MOUSTEST C

  Parameters
   (output)     buffer_size   Bytes for saving mouse state

  Returned      (function returns nothing)

  Variables     m2      Local variable for register bx

  Description   Determines the number of bytes required for
                saving the current state of the mouse
  ---------------------------------
*/

void mouse_storage( int *buffer_size )
{
  int m2,

  _asm
    {
    mov  ax, 21
    int  33h
    mov  m2, bx
    }
```

```
   *buffer_size = m2;
}


/* ---------------------------------
   Function:      mouse_save()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
    (in/out)      buffer    Buffer for saving mouse state

   Returned:      (function returns nothing)

   Variables:     buffer_seg     Segment of the buffer
                  buffer_off     Offset of the buffer

   Description:   Saves the current state of the mouse
   ---------------------------------
*/

void mouse_save( char far *buffer )
{
   unsigned buffer_seg = FP_SEG( buffer );
   unsigned buffer_off = FP_OFF( buffer );

   _asm
      {
      mov  ax, 22
      mov  es, buffer_seg
      mov  dx, buffer_off
      int  33h
      }
}


/* ---------------------------------
   Function:      mouse_restore()
   Toolbox:       MOUSEFUN.C
   Demonstrated:  MOUSTEST.C

   Parameters:
    (input)       buffer    Buffer for restoring the mouse state

   Returned:      (function returns nothing)

   Variables:     buffer_seg     Segment of the buffer
                  buffer_off     Offset of the buffer

   Description:   Restores the current state of the mouse
   ---------------------------------
*/

void mouse_restore( char far *buffer )
{
   unsigned buffer_seg = FP_SEG( buffer );
   unsigned buffer_off = FP_OFF( buffer );

   _asm
      {
      mov  ax, 23
      mov  es, buffer_seg
      mov  dx, buffer_off
      int  33h
      }
}


/* ---------------------------------
```

```
  Function:      mouse_setsensitivity()
  Toolbox:       MOUSEFUN.C
  Demonstrated:  MOUSTEST.C

  Parameters:
   (input)       horz      Relative horizontal sensitivity
   (input)       vert      Relative vertical sensitivity
   (input)       threshold Relative double speed threshold

  Returned:      (function returns nothing)

  Variables:     (none)

  Description:   Sets the mouse sensitivity and double speed
                 threshold
  ---------------------------------
*/

void mouse_setsensitivity( int horz, int vert, int threshold )
{
   _asm
      {
      mov  ax, 26
      mov  bx, horz
      mov  cx, vert
      mov  dx, threshold
      int  33h
      }
}


/* ---------------------------------
  Function:      mouse_getsensitivity()
  Toolbox:       MOUSEFUN.C
  Demonstrated:  MOUSTEST.C

  Parameters:
   (output)      horz      Relative horizontal sensitivity
   (output)      vert      Relative vertical sensitivity
   (output)      threshold Relative double speed threshold

  Returned:      (function returns nothing)

  Variables:     (none)

  Description:   Gets the mouse sensitivity and double speed
                 threshold
  ---------------------------------
*/

void mouse_getsensitivity( int *horz, int *vert, int *threshold )
{
   int m2, m3, m4;

   _asm
      {
      mov  ax, 27
      int  33h
      mov  m2, bx
      mov  m3, cx
      mov  m4, dx
      }

   *horz = m2;
   *vert = m3;
   *threshold = m4;
}
```

```
/* ---------------------------------
  Function:     mouse_setmaxrate()
  Toolbox:      MOUSEFUN.C
  Demonstrated: MOUSTEST.C

  Parameters:
   (input)      interrupts_per_second   Interrupt rate

  Returned:     (function returns nothing)

  Variables:    rate     Number for range of interrupt rates

  Description:  Sets the interrupt rate (InPort mouse only)
  ---------------------------------
*/

void mouse_setmaxrate( int interrupts_per_second )
{
   int rate;

   if ( interrupts_per_second <= 0 )
      rate = 0;
   else if ( interrupts_per_second > 0 && interrupts_per_second <= 30 )
      rate = 1;
   else if ( interrupts_per_second > 30 && interrupts_per_second <= 50 )
      rate = 2;
   else if ( interrupts_per_second > 50 && interrupts_per_second <= 100 )
      rate = 3;
   else
      rate = 4;

   _asm
      {
      mov  ax, 28
      mov  bx, rate
      int  33h
      }
}


/* ---------------------------------
  Function:     mouse_setpage()
  Toolbox:      MOUSEFUN.C
  Demonstrated: MOUSTEST.C

  Parameters:
   (input)      crt_page  Video page for mouse cursor

  Returned:     (function returns nothing)

  Variables:    (none)

  Description:  Sets the video page where mouse cursor appears
  ---------------------------------
*/

void mouse_setpage( int crt_page )
{
   _asm
      {
      mov  ax, 29
      mov  bx, crt_page
      int  33h
      }
}


/* ---------------------------------
  Function:     mouse_getpage()
```

```
  Toolbox:      MOUSEFUN.C
  Demonstrated: MOUSTEST.C

  Parameters:
   (output)    crt_page  Video page for mouse cursor

  Returned:     (function returns nothing)

  Variables:    m2        Local variable for register bx

  Description:  Gets the video page in which mouse cursor appears
  ---------------------------------
*/

void mouse_getpage( int *crt_page )
{
   int m2;

   _asm
      {
      mov  ax, 30
      int  33h
      mov  m2, bx
      }

   *crt_page = m2;
}


/* ---------------------------------
  Function:     mouse_setlang()
  Toolbox:      MOUSEFUN.C
  Demonstrated: MOUSTEST.C

  Parameters:
   (input)     language  Language number

  Returned:     (function returns nothing)

  Variables:    (none)

  Description:  Sets the language for mouse driver messages
  ---------------------------------
*/

void mouse_setlang( int language )
{
   _asm
      {
      mov  ax, 34
      mov  bx, language
      int  33h
      }
}


/* ---------------------------------
  Function:     mouse_getlang()
  Toolbox:      MOUSEFUN.C
  Demonstrated: MOUSTEST.C

  Parameters:
   (output)    language  Language number

  Returned:     (function returns nothing)

  Variables:    (none)

  Description:  Gets the language for mouse driver messages
```

```
  ---------------------------------
*/

void mouse_getlang( int *language )
{
   int m2;

   _asm
      {
      mov  ax, 35
      int  33h
      mov  m2, bx
      }

   *language = m2;
}


/* ---------------------------------
   Function:     mouse_getversion()
   Toolbox:      MOUSEFUN.C
   Demonstrated: MOUSTEST.C

   Parameters:
    (output)    version      Mouse driver version number
    (output)    mouse_type   Type of mouse
    (output)    irq_num      Interrupt request type

   Returned:     (function returns nothing)

   Variables:    m2      Local variable for register bx
                 m3      Local variable for register cx
                 maj     Major part of version number
                 min     Minor part of version number

   Description:  Gets the mouse driver version number, mouse type,
                 and interrupt request type
   ---------------------------------
*/

void mouse_getversion( double *version, int *mouse_type, int *irq_num )
{
   int m2, m3;
   int maj, min;

   _asm
      {
      mov  ax, 36
      int  33h
      mov  m2, bx
      mov  m3, cx
      }

   maj = ( m2 >> 12 ) * 10 + (( m2 >> 8 ) & 0xf );
   min = (( m2 >> 4 ) & 0xf ) * 10 + ( m2 & 0xf );
   *version = maj + min / 100.0;
   *mouse_type = m3 >> 8;
   *irq_num = m3 & 0xff;
}
```

```
/* ---------------------------------------------------------
   Name         ROTATE C
   Type         Test to collect data for thesis for Ron Archer
   Language     Microsoft QuickC version 2

   Program List  ROTATE C
                                          BOX C
                                          CLOCK.C
                                          DATA_PLT C
                                          DSK_INIT C
                                          EDIT C
                                          FILE C
                                          GETKEY C
                                          LIST C
                                          MENU C
                                          MOUSEFUN C
                                          MN_MENU C
                                          T_MANAGER C
                                          TEST_1 C
                                          T1OBJECT C
                                          SOUND C
                                          ST_MENU C
                                          STATS C
                                          VIDEO C

   Variables   head   global pointer to head of linked list
                          of student record indexes
               tail   global pointer to tail of linked list
                          of student record indexes
               r_head global pointer to head of linked list
                          of student data records
               r_tail global pointer to head of linked list
                          of student data records

   Usage       (no command line parameters)

   Description  Computer based test that measures an individual's
                     abilities in very specific areas

          Last Revision  10 March 1996  Animesh Banerjee
   --------------------------------------------------------- */


#include <stdio.h>
#include <graph.h>
#include <dos.h>
#include "typ_init.h"
#include "getkey.h"
#include "menu.h"
#include "box.h"
#include "t_colors.h"
#include "list.h"
#include "mn_menu.h"

char *school_info[] =
  {
  " ",
          "       Mental Rotation Test ",
  " ",
          "            Ver  1.00",
  " ",
          "   Embry-Riddle Aeronautical University ",
  " ",
  "< Press any key >",
  NULL
  },

char *my_info[] =
```

```
{
" ",
"      Mental Rotation Test ",
" ",
"          Version 1.00 ",
" ",
" ",
"            by ",
" ",
" ",
"       Ronald D. Archer ",
" ",
" Embry-Riddle Aeronautical University ",
"      Daytona Beach, FL 32114 ",
" ",
"          Tel: (904) 322-5501 ",
" banerjea@erau.db.erau.edu (internet) ",
" ",
"< Press any key >",
NULL
};

int  q_in_record = 1;

NODE    *head, *tail;
RES_NODE *r_head, *r_tail;


void main( void )
{
  int *save_info_box;

  /* Initialize text foreground and background color */
  _settextcolor( T_BLUE );
  _setbkcolor( BK_BLACK );

  /* Initialize video */
  _setvideomode( _TEXTC80 );
  _clearscreen( _GCLEARSCREEN );

  /* Display school information message */
  save_info_box = menu_message( 7, 18, school_info );

  /* get key or mouse press */
  getkey_or_mouse();

  /* Erase school information message */
  menu_erase( save_info_box );

  /* Display my information message */
          save_info_box = menu_message( 4, 18, my_info );

  /* get key or mouse press */
  getkey_or_mouse();

  /* Erase school information message */
  menu_erase( save_info_box );

  /* Set foreground and background colors for program */
  _setbkcolor( BK_CYAN );
  _settextcolor( T_BLACK );

  /* Fill the background */

   box_charfill( 1, 1, 25, 80, 178 );

  /* activate main menu */
  display_main_menu( &head, &tail, &r_head, &r_tail );}
```

```
/* ---------------------------------------------
   Name          SOUND C
   Type          Toolbox module
   Language      Microsoft QuickC
   Demonstrated  SOUNTEST C
   Video         (no special video requirements)
   ---------------------------------------------
*/

#include <conio h>
#include <time h>
#include "sound h"

static unsigned control,
static int control_flag = 1,

/* ---------------------------------
   Function      speaker_toggle()
   Toolbox       SOUND C
   Demonstrated  SOUNTEST C

   Parameters    (none)

   Returned      (function returns nothing)

   Variables     (none)

   Description   Pulses the speaker on or off with each call
   ---------------------------------
*/

void speaker_toggle( void )
{
   if ( control_flag )
      {
      control = inp( 0x61 ),
      control_flag = 0,
      }
   outp( 0x61, ( inp( 0x61 ) & 0xFE ) ^ 2 ),
}


/* ---------------------------------
   Function      sound()
   Toolbox       SOUND C
   Demonstrated  SOUNTEST C

   Parameters
    (input)      frequency     Frequency of generated tone

   Returned      (function returns nothing)

   Variables     divisor       Timer value for given frequency

   Description   Sets a tone at a given frequency
   ---------------------------------
*/

void sound( int frequency )
{
   unsigned divisor,

   divisor = (unsigned)( 1193180L / frequency ),
   if ( control_flag )
      {
      outp( 0x43, 0xB6 ),
      outp( 0x42, divisor % 256 ),
      outp( 0x42, divisor / 256 ),
      control = inp( 0x61 ),
```

```
      control_flag = 0;
      }
   else
      {
      divisor = (unsigned)( 1193180L / frequency );
      outp( 0x42, divisor % 256 );
      outp( 0x42, divisor / 256 );
      }
   outp( 0x61, control | 3 );
}


/* ---------------------------------
   Function:      silence()
   Toolbox:       SOUND.C
   Demonstrated:  SOUNTEST.C

   Parameters:    (none)

   Returned:      (function returns nothing)

   Variables:     (none)

   Description:   Turns off the tone generator
   ---------------------------------
*/

void silence( void )
{
   outp( 0x61, control );
   control_flag = 1;
}


/* ---------------------------------
   Function:      wait_ticks()
   Toolbox:       SOUND.C
   Demonstrated:  SOUNTEST.C

   Parameters:
    (input)       ticks          Number of clock ticks

   Returned:      (function returns nothing)

   Variables:     now            Time as returned by sound()

   Description:   Delays for a given number of clock ticks
   ---------------------------------
*/

void wait_ticks( unsigned ticks )
{
   clock_t now;

   do
      {
      now = clock();
      while ( clock() == now )
         {;}
      }
   while( --ticks );
}


/* ---------------------------------
   Function:      warble()
   Toolbox:       SOUND.C
   Demonstrated:  SOUNTEST.C
```

```
  Parameters
   (input)     count    Number of warble cycles

  Returned     (function returns nothing)

  Variables    (none)

  Description  Creates a three-tone warble
  ----------------------------------
*/

void warble( int count )
{
   do
      {
      sound( 500 ),
      wait_ticks( 1 ),
      sound( 2000 ),
      wait_ticks( 1 ),
      sound( 1000 ),
      wait_ticks( 1 ),
      sound( 750 ),
      wait_ticks( 1 ),
      }
   while ( --count ),

   silence(),
}


/* ----------------------------------
  Function     weird()
  Toolbox      SOUND C
  Demonstrated  SOUNTEST C

  Parameters    count    Number of sound generation cycles

  Returned     (function returns nothing)

  Variables    i        Looping index
               j        Tone frequency

  Description  Creates a modulated sound
  ----------------------------------
*/

void weird( int count )
{
   int i, j,

   sound( 50 ),
   do
      for ( i = 50, i < 1200, i += 100 )
         for ( j = i, j < i + 1200, j += 5 )
            sound( j ),
   while ( --count ),

   silence(),
}


/* ----------------------------------
  Function     siren()
  Toolbox      SOUND C
  Demonstrated  SOUNTEST C

  Parameters    count    Number of sound generation cycles

  Returned     (function returns nothing)
```

```
  Variables:    i       Looping index

  Description:  Creates a sound whose frequency rises and falls
  ---------------------------------
*/

void siren( int count )
{
   int i;

   sound( 50 );
   do
      {
      for ( i = 50; i < 2000; i++ )
         sound( i );
      for ( i = 2000; i > 50; i-- )
         sound( i );
      }
   while ( --count );

   silence();
}


/* ---------------------------------
  Function:     white_noise()
  Toolbox:      SOUND.C
  Demonstrated: SOUNTEST.C

  Parameters:   ticks   Number of clock ticks

  Returned:     (function returns nothing)

  Variables:    i       Looping index
                rndm    Pseudorandom unsigned integer
                now     Time as returned by clock()

  Description:  Generates white noise, a wide_ranging multifrequency
                sound
  ---------------------------------
*/

void white_noise( int ticks )
{
   unsigned i, rndm;
   clock_t now;

   do
      {
      now = clock();
      while ( clock() == now )
         {
         speaker_toggle();
         rndm = rndm * 317 + 21317;
         for ( i = rndm & 0xFF; i; i-- )
            {;}
         }
      }
   while( --ticks );

   silence();
}


/* ---------------------------------
  Function:     note()
  Toolbox:      SOUND.C
  Demonstrated: SOUNTEST.C
```

```
  Parameters:    frequency     Frequency of the tone
                 ticks         Number of clock ticks

  Returned:      (function returns nothing)

  Variables:     (none)

  Description:   Creates a tone given its frequency and duration
  ---------------------------------
*/

void note( int frequency, int ticks )
{
   sound( frequency );
   wait_ticks( ticks );
   silence();
}
```

```
/* ---------------------------------------------------
  Name:         STATS.C
  Type:         Routines for statistical analysis
                of student results
                Air Traffic Control Screening Program
  Language:     Microsoft QuickC version 2
  ---------------------------------------------------
*/

#include <stdio.h>
#include <math.h>
#include <float.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include "typ_init.h"
#include "menu.h"
#include "getkey.h"
#include "box.h"
#include "t_colors.h"
#include "sound.h"


/* Error message data */
char *error_box_3_01[] =
  {
  " Error Message #3.01 ",
  "",
  " One of the statistical functions ",
  " was passed a value for test_no ",
  " which is out of range ",
  "",
  "< Press any key >",
  NULL
  };

char *error_box_3_02[] =
  {
  " Error Message #3.02 ",
  "",
  " There are no student records in ",
  " memory to analyze. ",
  "",
  " RESULT => No statistical analysis ",
  " of student results can be done. ",
  "",
  " < Press any key >",
  NULL
  };

/*
  this function calculates the mean time
  for correct answers for test number
  determined by test_no
*/

double cal_mean_time_correct( int test_no, RES_NODE *h )
{
  double sum_time = 0.0;
  int num_students;
  int *save_error_box;

  RES_NODE *n;

  /*
    check to see if test_no in range
  */
  if ( test_no < 0 || test_no > 19 ) {
    /*
```

```
      set error box color to red
      set error text color to white
    */
    menu_back_color( BK_RED ),
    menu_text_color( T_WHITE | T_BRIGHT ),

    /* Display error_box_3_01 */
    save_error_box = menu_message( 10, 8, error_box_3_01 )

    getkey_or_mouse(),

    /* Erase error_box_3_01 */
    menu_erase( save_error_box ),

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE ),
    menu_text_color( T_BLACK ),

    return( (double) 0 0 ),          /* error return 0 to caller    */
  }

  n = h,                          /* set pointer to head of list  */
  while ( n != NULL ) {           /* while not end of list        */
    /*
      increase total time by avg time correct
      for this student for this test number
    */
    sum_time = sum_time + n->student_info[test_no] avg_time_correct,

    /*
      increase number of students
      results taken from
    */
    ++num_students,

    n = n->next,
  }

  /*
    return the mean time for correct answer times
    for test test_no
  */
  if ( num_students == 0 )
    /*
      check for divide by zero error
    */
    return( (double) 0 ),
  else
    return ( sum_time / (double) num_students ),
}


/*
  this function calculates the mean time
  for incorrect answers for test number
  determined by test_no
*/

double cal_mean_time_incorrect( int test_no  RES_NODE *h )
{
  double sum_time,
  int num_students,
  int *save_error_box,

  RES_NODE *n,
```

```
  Description:  This function calculates the statistical
                deviation for correct answers for test number
                determined by test_no

  --------------------------------
*/

double cal_stat_deviation_correct( int test_no, RES_NODE *h )
{
  double sum_difference,
  double difference,
  double mean_time_correct,
  int *save_error_box,

  RES_NODE *n,

  /*
    check to see if test_no in range
  */
  if ( test_no < 0 || test_no > 19 ) {
    /*
      set error box color to red
      set error text color to white
    */
    menu_back_color( BK_RED ),
    menu_text_color( T_WHITE | T_BRIGHT ),

    /* Display error_box_3_01 */
    save_error_box = menu_message( 10, 8, error_box_3_01 ),

    getch(),

    /* Erase error_box_3_01 */
    menu_erase( save_error_box ),

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE ),
    menu_text_color( T_BLACK ),

    return( (double) 0 ),              /* error return 0 to caller     */
  }

  /*
    get the mean response time
    for correct answers
  */
  mean_time_correct = cal_mean_time_correct( test_no, h ),

  n = h,                             /* set pointer to head of list  */
  while ( n != NULL ) {              /* while not end of list        */
    /*
      calculate difference from mean
    */
    difference = mean_time_correct -
              n->student_info[test_no] avg_time_correct,

    /*
      square difference
    */
    difference = difference * difference,

    /*
      update sum of difference
    */
```

```
    sum_difference = sum_difference + difference,

    n = n->next,
  }

  /*
    return the statistical deviation for correct answer times
    for test test_no
  */
    return ( sqrt( (double) sum_difference ) ),
}


/*
  this function calculates the statistical
  deviation for incorrect answers for test number
  determined by test_no
*/

double cal_stat_deviation_incorrect( int test_no, RES_NODE *h )
{
  double sum_difference,
  double difference,
  double mean_time_correct,
  int *save_error_box;

  RES_NODE *n,

  /*
    check to see if test_no in range
  */
  if ( test_no < 0 || test_no > 19 ) {
    /*
      set error box color to red
      set error text color to white
    */
    menu_back_color( BK_RED ),
    menu_text_color( T_WHITE | T_BRIGHT ),

    /* Display error_box_3_01 */
    save_error_box = menu_message( 10, 8, error_box_3_01 ),

    getch();

    /* Erase error_box_3_01 */
    menu_erase( save_error_box ),

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE ),
    menu_text_color( T_BLACK ),

    return( (double) 0 ),              /* error return 0 to caller    */
  }

  /*
    get the mean response time
    for correct answers
  */
  mean_time_correct = cal_mean_time_incorrect( test_no, h ),

  n = h,                           /* set pointer to head of list  */
  while ( n != NULL ) {              /* while not end of list       */
    /*
      calculate difference from mean
    */
    difference = mean_time_correct -
```

```
            n->student_info[test_no] avg_time_incorrect,

    /*
      square difference
    */
    difference = difference * difference,

    /*
      update sum of difference
    */
    sum_difference = sum_difference + difference,

    n = n->next,
  }

  /*
    return the statistical deviation for correct answer times
    for test test_no
  */
    return ( sqrt( (double) sum_difference ) ),
}


/* ---------------------------------
  Function      Stats_test_1(),
  File          STATS C

  Parameters    None

  Returned      None

  Variables     None

  Description   Calculates statistics for test #1 given results
                from the test
  ---------------------------------
*/

void stats_test_1( TEMP *st1, STUDENT_RECORD *new_student, int *correct,
                   int test_num )
{

  int n,
  int tot_num_correct = 0,
  int tot_num_incorrect = 0,
  double tot_time_incorrect = 0 0,
  double tot_time_correct =  0 0,
  char lloop_limit, uloop_limit,
  int sum_correct = 0,
  int sum_incorrect = 0,
  double sum_time_correct = 0 0,
  double sum_time_incorrect = 0 0,

  /* check trial number to set corresponding loop counters */
  if ( test_num == 0 )
  {
          lloop_limit = 0,
          uloop_limit = 32,
  }
  else
  {
          lloop_limit = 32,
          uloop_limit = 64,
  }

  /* begin processing of data */

  for ( n = lloop_limit, n < uloop_limit, n++ )
  {
```

```
  if ( st1[n] answer == correct[n] ) {

    ++sum_correct,
    sum_time_correct = sum_time_correct +
                                ( st1[n] reaction_time / CLK_TCK ),
                                st1[n] right_wrong = 1,
    }

  else {

    ++sum_incorrect,
    sum_time_incorrect = sum_time_incorrect +
                                ( st1[n] reaction_time / CLK_TCK ),
                                st1[n] right_wrong = 0,
    }

}
/* get number of questions */
        new_student->student_info[test_num] total_no_questions = 64,

/* check for divide by zero */
if ( sum_correct != 0 )

  /* calculate average time to answer questions correctly for trial */
                new_student->student_info[test_num] avg_time_correct =
          sum_time_correct / ( double ) sum_correct,

else

  /* calculate average time to answer questions for trial correctly */
                new_student->student_info[test_num] avg_time_correct = 0 0,

/* check for divide by zero */
if ( sum_incorrect != 0 )

  /* calculate average time to answer questions for trial incorrectly */
                new_student->student_info[test_num] avg_time_incorrect =
          sum_time_incorrect / ( double ) sum_incorrect,

else

  /* calculate average time to answer questions incorrectly for the trial*/
                new_student->student_info[test_num] avg_time_incorrect = 0 0,

/* get number of questions answered correctly for trial
          NOTE  Score is (number correct - number incorrect)  */
        new_student->student_info[test_num] no_questions_correct =
                                                sum_correct - sum_incorrect,

/* Calculate the overall average incorrect and correct times for all 64 questions */
for( n = 0, n < 64, n++ )
{
        if( st1[n] right_wrong == 1 )
        {
                tot_time_correct += st1[n] reaction_time / CLK_TCK,
                tot_num_correct++,
        }
        else
        {
                tot_time_incorrect += st1[n] reaction_time / CLK_TCK,
                tot_num_incorrect++,
        }
}

if ( tot_num_correct == 0 )
                new_student->student_info[1] ovrl_avg_time_corr = 0 0,
else
                new_student->student_info[1] ovrl_avg_time_corr =
```

```
                                        tot_time_correct / (double) tot_num_correct,

  if ( tot_num_incorrect == 0 )
                        new_student->student_info[1] ovrl_avg_time_incorr = 0 0,
  else
                        new_student->student_info[1] ovrl_avg_time_incorr =
                                        tot_time_incorrect / (double) tot_num_incorrect,


  for( n = 0, n < 64, n++ )
  { new_student->RESPONSE[n] reaction_time = st1[n] reaction_time/CLK_TCK,
    new_student->RESPONSE[n] answer = st1[n] answer,
    new_student->RESPONSE[n] right_wrong = st1[n] right_wrong,
  }

  /*
     update student record to indicate that student has
     accomplished test #1
  */
            new_student->test_no = test_num
}

/* ---------------------------------
   Function       Stats_test_2(),
   File           STATS C

   Parameters     None

   Returned       None

   Variables      None

   Description    Calculates statistics for test #2 given results
                                        from the test
   ---------------------------------
*/

void stats_test_2( TEMP *st1, STUDENT_RECORD *new_student, int *correct,
                                        int test_num )
{
  int n,
  int tot_num_correct = 0,
  int tot_num_incorrect = 0,
  double tot_time_incorrect = 0 0,
  double tot_time_correct =  0 0,
  char  uloop_limit, lloop_limit,
  int sum_correct = 0,
  int sum_incorrect = 0,
  double sum_time_correct = 0 0,
  double sum_time_incorrect = 0 0,

  /* check trial number to set corresponding loop counters */
  if ( test_num == 0 )
  {
            lloop_limit = 0,
            uloop_limit = 33,
  }
  else
  {
            lloop_limit = 33
            uloop_limit = 65,
  }

  /* begin processing of data */

  for ( n = lloop_limit, n < uloop_limit, n++ )
  {

             if ( st1[n] answer == correct[n] ) {
```

```
                ++sum_correct,
                sum_time_correct = sum_time_correct +
                                ( st1[n] reaction_time / CLK_TCK ),
                                st1[n] right_wrong = 1,
                }

        else {

                ++sum_incorrect,
                sum_time_incorrect = sum_time_incorrect +
                                ( st1[n] reaction_time / CLK_TCK ),
                                st1[n] right_wrong = 0,
                }

}
/*
        * get number of questions
        * +2 here used to allow for space taken up by test #1
        */
new_student->student_info[test_num + 2] total_no_questions = 65,

/* check for divide by zero */
if ( sum_correct != 0 )

        /* calculate average time to answer questions correctly for trial */
                new_student->student_info[test_num + 2] avg_time_correct =
                                sum_time_correct / ( double ) sum_correct,

else

        /* calculate average time to answer questions for trial correctly */
                new_student->student_info[test_num + 2] avg_time_correct = 0 0

/* check for divide by zero */
if ( sum_incorrect != 0 )

        /* calculate average time to answer questions for trial incorrectly */
                new_student->student_info[test_num + 2] avg_time_incorrect =
                                sum_time_incorrect / ( double ) sum_incorrect,

else

        /* calculate average time to answer questions incorrectly for the trial*/
                new_student->student_info[test_num + 2] avg_time_incorrect = 0 0,

/* get number of questions answered correctly for trial
*/
        new_student->student_info[test_num + 2] no_questions_correct = sum_correct,

/* Calculate the overall average incorrect and correct times for all questions */
for( n = 0, n < 65, n++ )
{
        if( st1[n] right_wrong == 1 )
        {
                tot_time_correct += st1[n] reaction_time / CLK_TCK,
                tot_num_correct++,
        }
        else
        {
                tot_time_incorrect += st1[n] reaction_time / CLK_TCK,
                tot_num_incorrect++,
        }
}

if ( tot_num_correct == 0 )
                new_student->student_info[2] ovrl_avg_time_corr = 0 0
else
                new_student->student_info[2] ovrl_avg_time_corr =
```

```
                                        tot_time_correct / (double) tot_num_correct;

  if ( tot_num_incorrect == 0 )
                        new_student->student_info[2].ovrl_avg_time_incorr = 0.0;
  else
                        new_student->student_info[2].ovrl_avg_time_incorr =
                                        tot_time_incorrect / (double) tot_num_incorrect;

  for( n = 0; n < 65; n++ )  /* 23 is offset for # problems in test 1 */
  {
            new_student->RESPONSE[n+23].reaction_time = st1[n].reaction_time/CLK_TCK;
            new_student->RESPONSE[n+23].answer = st1[n].answer;
            new_student->RESPONSE[n+23].right_wrong = st1[n].right_wrong;
  }


  /*
            update student record to indicate that student has
            accomplished test #2
  */
          new_student->test_no = test_num;
}

/* ---------------------------------
  Function:     Get_mtc_data();
  File:         STATS.C

  Parameters:
    (input)     value     array of tyoe float holding values for
                          mean time for correct answer for each
                          student

  Returned:     None

  Variables:    n         Pointer to node of type RES_NODE

  Description:  Get mean time for correct answer data and
                place it into array value

  ---------------------------------
*/

void Get_mtc_data( float *value, RES_NODE *h )
{
  int counter;
  int *save_error_box;
  RES_NODE *n; n = h;

  /*
    check to see if any student records
    in linked list
  */
  if ( n == NULL ) {
    /*
      set error box color to red
      set error text color to white
    */
    menu_back_color( BK_RED );
    menu_text_color( T_WHITE | T_BRIGHT );

    /* Display error_box_3_02 */
    save_error_box = menu_message( 13, 19, error_box_3_02 );

    /* Make error sound */
    warble( 5 );

    getkey_or_mouse();

    /* Erase error_box_3_02 */
```

```
    menu_erase( save_error_box ),

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE ),
    menu_text_color( T_BLACK ),
    }
  else {
    for ( counter = 0, counter <= n->test_no, counter++ ) {
      *value = (float) n->student_info[counter] avg_time_correct,

      /* advance pointer to next array location */
      ++value,
    }
  }
}


/* ---------------------------------
  Function      Stats_test_3(),
  File          STATS C

  Parameters    None

  Returned      None

  Variables     None

  Description   Calculates statistics for test #3 given results
                from the test
  ---------------------------------
*/

void stats_test_3( TEMP *st1, STUDENT_RECORD *new_student, char *correct[],
                int problems, int trial_num )
{
  int n,
  int present,
  int sum_correct = 0,
  int sum_incorrect = 0,
  double sum_time_correct = 0 0,
  double sum_time_incorrect = 0 0,

            for ( n = 0, n < (problems * 3), n++ ) {
    /* get digit and convert to integer */
    present = atoi( correct[n] ) + 48,

    /* did student answer correctly */
    if ( st1[n] answer == present ) {

      /* student answered correctly */
      ++sum_correct,
      sum_time_correct = sum_time_correct +
                                                                    ( st1[n] reaction_time /
CLK_TCK ),
      }

    else {

      /* student answered incorrectly */
      ++sum_incorrect,
      sum_time_incorrect = sum_time_incorrect +

st1[n] reaction_time / CLK_TCK ),
      }
    /* move pointer to next result */
    /* ++correct, */
```

```
  }
  /* get number of questions */
          new_student->student_info[trial_num + 16] total_no_questions = problems * 3,

  /* check for divide by zero */
  if ( sum_correct != 0 )

    /* calculate average time to answer questions correctly */
                  new_student->student_info[trial_num + 16] avg_time_correct =
          sum_time_correct / ( double ) sum_correct,
  else

    /* calculate average time to answer questions correctly */
                  new_student->student_info[trial_num + 16] avg_time_correct = 0 0,


  /* check for divide by zero */
  if ( sum_incorrect != 0 )

    /* calculate average time to answer questions incorrectly */
                  new_student->student_info[trial_num + 16] avg_time_incorrect =
        sum_time_incorrect / ( double ) sum_incorrect,
  else

    /* calculate average time to answer questions incorrectly */
                  new_student->student_info[trial_num + 16] avg_time_incorrect = 0 0,

  /* get number of questions answered correctly */
          new_student->student_info[trial_num + 16] no_questions_correct = sum_correct,

  /*
    update student record to indicate that student has
    accomplished test #3 trial #trial_num
  */
          new_student->test_no = trial_num + 16,
}


/* ---------------------------------
  Function      Get_mti_data(),
  File          STATS.C

  Parameters
    (input)     value     array of tyoe float holding values for
                          mean time for correct answer for each
                          student

  Returned      None

  Variables     n         Pointer to node of type RES_NODE

  Description   Get mean time for incorrect answer data and
                place it into array value

  ---------------------------------
*/

void Get_mti_data( float *value, RES_NODE *h )
{
  int counter,
  int *save_error_box,
  RES_NODE *n, n = h,

  /*
    check to see if any student records
    in linked list
  */
  if ( n == NULL ) {
```

```
    /*
      set error box color to red
      set error text color to white
    */
    menu_back_color( BK_RED );
    menu_text_color( T_WHITE | T_BRIGHT );

    /* Display error_box_3_02 */
    save_error_box = menu_message( 13, 19, error_box_3_02 );

    /* Make error sound */
    warble( 5 );

    getkey_or_mouse();

    /* Erase error_box_3_02 */
    menu_erase( save_error_box );

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE );
    menu_text_color( T_BLACK );
    }

  else {

    for ( counter = 0; counter <= n->test_no; counter++ ) {

      *value = (float) n->student_info[counter].avg_time_incorrect;

      /* advance pointer to next array location */
      ++value;
    }
  }
}


/* ---------------------------------
  Function:     Get_pc_data();
  File:         STATS.C

  Parameters:
    (input)     value     array of tyoe float holding values for
                          average percentage correct for all
                          students.

  Returned:     None

  Variables:    n         Pointer to node of type RES_NODE

  Description:  Get percentage of correct answers for each
                trial, and place it into array value
  ---------------------------------
*/
void Get_pc_data( float *value, RES_NODE *h )
{
  int counter;
  int *save_error_box;
  RES_NODE *n; n = h;

  /*
    check to see if any student records
    in linked list
  */
  if ( n == NULL ) {
    /*
```

```
      set error box color to red
      set error text color to white
    */
    menu_back_color( BK_RED );
    menu_text_color( T_WHITE | T_BRIGHT );

    /* Display error_box_3_02 */
    save_error_box = menu_message( 13, 19, error_box_3_02 );

    /* Make error sound */
    warble( 5 );

    getkey_or_mouse();

    /* Erase error_box_3_02 */
    menu_erase( save_error_box );

    /*
      set box color back to cyan
      set text color back to black
    */
    menu_back_color( BK_WHITE );
    menu_text_color( T_BLACK );
    }
  else {

    for ( counter = 0; counter <= n->test_no; counter++ ) {

      if ( n->student_info[counter].total_no_questions >= 1 )

        *value = ( (float) n->student_info[counter].no_questions_correct /
              (float) n->student_info[counter].total_no_questions )
              * 100.0 ;
      else
        *value = 0.0;

      /* advance pointer to next array location */
      ++value;
    }
  }
/* ---------------------------------
  Function:     Mean_time_correct();
  File:         STATS.C

  Parameters:
    (input)     value     array of tyoe float holding values for
                          average percentage correct for all
                          students.

  Returned:     None

  Variables:    n         Pointer to node of type RES_NODE

  Description:  Get percentage of correct answers for each
                trial, and place it into array value

  ---------------------------------
*/
void mean_time_correct( float *value, RES_NODE *h )
{
  int counter;

  for ( counter = 0; counter <= 19; counter++ ) {

      *value = cal_mean_time_correct( counter, h );

      /* advance pointer to next array location */
      ++value;
  }
```

```
/* ---------------------------------------------
  Name        T1OBJECTS C
  Type        Routines to implement graphic objects that are used
                                        in the test and other utilities in battery
                                        Airport Security Personnel Screening Program
  Language    Microsoft QuickC version 2
  ---------------------------------------------
*/


#include <graph h>
#include <math h>
#include <malloc h>
#include <conio h>
#include <stdio h>
#include "video h"
#include "t_colors h"
#include "sound h"
#include "t1object h"
#include "video h"

/* set number of problems in test */
#define NUM_PROBLEMS  64  /* NOTE  this parameter is also defined in test_1 c */

/*
        Declare global pointers to objects to be drawn on screen
*/
/* pointers to buffers holding images of all possible orientations of the aircraft */

  char *aircraft_ptr[8],

/* ---------------------------------
  Function    Draw_background(),

  File        TEST_1 C

  Parameters   None

  Returned     None

  Description  Draws 8 white solid circles, on the circumference of
                                        a larger circle (not drawn), each 45 degrees apart
                                        from each other with respect to the center of the
                                        screen  A solid white triangle is drawn in the center
                                        of the screen as well

---------------------------------
*/

void Draw_background( void )
{
        int del_x = 2, del_y = 2,

        int p1_x= 392, p1_y=295, p2_x=408, p2_y=295, p3_x=400, p3_y=306,

        int c0_b1_x = 150,  c0_b1_y = 50, c0_b2_x = 650,  c0_b2_y = 550,


                c1_b1_x = 650 + del_x,  c1_b1_y = 300 + del_y,
                c1_b2_x = 650 - del_x,  c1_b2_y = 300 - del_y,


                c2_b1_x = 575 + del_x,  c2_b1_y = 475 + del_y,
                c2_b2_x = 575 - del_x,  c2_b2_y = 475 - del_y,


                c3_b1_x = 400 + del_x,  c3_b1_y = 550 + del_y,
                c3_b2_x = 400 - del_x,  c3_b2_y = 550 - del_y,
```

```
                c4_b1_x = 225 + del_x,  c4_b1_y = 475  + del_y,
                c4_b2_x = 225 - del_x,  c4_b2_y = 475 - del_y,


                c5_b1_x = 150 + del_x,  c5_b1_y = 300 + del_y,
                c5_b2_x = 150 - del_x,  c5_b2_y = 300 - del_y,


                c6_b1_x = 225 + del_x,  c6_b1_y = 125  + del_y,
                c6_b2_x = 225 - del_x,  c6_b2_y = 125 - del_y,


                c7_b1_x = 400 + del_x,  c7_b1_y =  50 + del_y,
                c7_b2_x = 400 - del_x,  c7_b2_y =  50 - del_y,


                c8_b1_x = 575 + del_x,  c8_b1_y = 125  + del_y,
                c8_b2_x = 575 - del_x,  c8_b2_y = 125 - del_y,


        /*_ellipse( _GBORDER , device_x(c0_b1_x), device_y(c0_b1_y),
                                                        device_x(c0_b2_x), device_y(c0_b2_y) ),
        */
        _ellipse( _GFILLINTERIOR , device_x(c1_b1_x), device_y(c1_b1_y),
                                                        device_x(c1_b2_x), device_y(c1_b2_y) ),
        _ellipse( _GFILLINTERIOR , device_x(c2_b1_x), device_y(c2_b1_y),
                                                        device_x(c2_b2_x), device_y(c2_b2_y) ),
        _ellipse( _GFILLINTERIOR , device_x(c3_b1_x), device_y(c3_b1_y),
                                                        device_x(c3_b2_x), device_y(c3_b2_y) ),
        _ellipse( _GFILLINTERIOR , device_x(c4_b1_x), device_y(c4_b1_y),
                                                        device_x(c4_b2_x), device_y(c4_b2_y) ),
        _ellipse( _GFILLINTERIOR , device_x(c5_b1_x), device_y(c5_b1_y),
                                                        device_x(c5_b2_x), device_y(c5_b2_y) ),
        _ellipse( _GFILLINTERIOR , device_x(c6_b1_x), device_y(c6_b1_y),
                                                        device_x(c6_b2_x), device_y(c6_b2_y) ),
        _ellipse( _GFILLINTERIOR , device_x(c7_b1_x), device_y(c7_b1_y),
                                                        device_x(c7_b2_x), device_y(c7_b2_y) ),
        _ellipse( _GFILLINTERIOR , device_x(c8_b1_x), device_y(c8_b1_y),
                                                        device_x(c8_b2_x), device_y(c8_b2_y) ),


        triangle(SOLID, device_x(p1_x), device_y(p1_y),
                                                device_x(p2_x), device_y(p2_y),
                                                device_x(p3_x), device_y(p3_y)),

}

/* ---------------------------------
  Function      Draw_example_background(),

  File          TEST_1 C

  Parameters    None

  Returned      None

  Description   Draws 8 white solid circles, on the circumference of
                                        a larger circle (not drawn), each 45 degrees apart
                                        from each other with respect to the center of the
                                        screen  A solid white triangle is drawn in the center
                                        of the screen as well
 ---------------------------------
*/

void Draw_example_background( void )
{
        int del_x = 2, del_y = 2,

        int p1_x= 392, p1_y=370, p2_x=408, p2_y=370, p3_x=400, p3_y=381,
```

```
int c0_b1_x = 200,  c0_b1_y = 175, c0_b2_x = 600,  c0_b2_y = 575,

        c1_b1_x = 600 + del_x,  c1_b1_y = 375 + del_y,
        c1_b2_x = 600 - del_x,  c1_b2_y = 375 - del_y,

        c2_b1_x = 541 + del_x,  c2_b1_y = 516 + del_y,
        c2_b2_x = 541 - del_x,  c2_b2_y = 516 - del_y,

        c3_b1_x = 400 + del_x,  c3_b1_y = 600 + del_y,
        c3_b2_x = 400 - del_x,  c3_b2_y = 600 - del_y,

        c4_b1_x = 259 + del_x,  c4_b1_y = 516  + del_y,
        c4_b2_x = 259 - del_x,  c4_b2_y = 516 - del_y,

        c5_b1_x = 200 + del_x,  c5_b1_y = 375 + del_y,
        c5_b2_x = 200 - del_x,  c5_b2_y = 375 - del_y,

        c6_b1_x = 259 + del_x,  c6_b1_y = 234 + del_y,
        c6_b2_x = 259 - del_x,  c6_b2_y = 234 - del_y,

        c7_b1_x = 400 + del_x,  c7_b1_y = 175 + del_y,
        c7_b2_x = 400 - del_x,  c7_b2_y = 175 - del_y,

        c8_b1_x = 541 + del_x,  c8_b1_y = 234 + del_y,
        c8_b2_x = 541 - del_x,  c8_b2_y = 234 - del_y,

    /*_ellipse( _GBORDER , device_x(c0_b1_x), device_y(c0_b1_y),
                                                device_x(c0_b2_x), device_y(c0_b2_y) ),
     */
    _ellipse( _GFILLINTERIOR , device_x(c1_b1_x), device_y(c1_b1_y),
                                                device_x(c1_b2_x), device_y(c1_b2_y) ),
    _ellipse( _GFILLINTERIOR , device_x(c2_b1_x), device_y(c2_b1_y),
                                                device_x(c2_b2_x), device_y(c2_b2_y) ),
    _ellipse( _GFILLINTERIOR , device_x(c3_b1_x), device_y(c3_b1_y),
                                                device_x(c3_b2_x), device_y(c3_b2_y) ),
    _ellipse( _GFILLINTERIOR , device_x(c4_b1_x), device_y(c4_b1_y),
                                                device_x(c4_b2_x), device_y(c4_b2_y) ),
    _ellipse( _GFILLINTERIOR , device_x(c5_b1_x), device_y(c5_b1_y),
                                                device_x(c5_b2_x), device_y(c5_b2_y) ),
    _ellipse( _GFILLINTERIOR , device_x(c6_b1_x), device_y(c6_b1_y),
                                                device_x(c6_b2_x), device_y(c6_b2_y) ),
    _ellipse( _GFILLINTERIOR , device_x(c7_b1_x), device_y(c7_b1_y),
                                                device_x(c7_b2_x), device_y(c7_b2_y) ),
    _ellipse( _GFILLINTERIOR , device_x(c8_b1_x), device_y(c8_b1_y),
                                                device_x(c8_b2_x), device_y(c8_b2_y) ),


    triangle(SOLID, device_x(p1_x), device_y(p1_y),
                    device_x(p2_x), device_y(p2_y),
                    device_x(p3_x), device_y(p3_y)),

}

/* ---------------------------------
  Function     Draw_plane(),

  File         TEST_1 C

  Parameters   float heading (in degrees)
```

```
  Returned:     None

  Description:  Draws a symbol for an airplane at a specified heading.

 ----------------------------------
*/
void Draw_plane( float heading )
{
        float x[6], y[6], x_set[6], y_set[6];
        int i;
        short previous;
        double theta;

        x_set[0]= 32.0;   y_set[0]= 48.0;
        x_set[1]= 32.0;   y_set[1]=  2.0;
        x_set[2]= 50.0;   y_set[2]= 25.0;
        x_set[3]=  0.0;   y_set[3]= 25.0;
        x_set[4]=  8.0;   y_set[4]= 34.0;
        x_set[5]=  8.0;   y_set[5]= 15.0;

        /* use rotation matrix to rotate points about center of picture (25,25)
         */

        /* convert heading to radians measured from horizontal x-axis*/
        theta = (double)(90.0 - heading) * 3.1415926536/180.0;

        for( i=0; i < 6; i++ )
        {
                x[i] = (float)cos(theta)*x_set[i] - (float)sin(theta)*y_set[i] + 25*(1-(float)cos(theta)) + 25*(float)sin(theta);
                y[i] = (float)sin(theta)*x_set[i] + (float)cos(theta)*y_set[i] + 25*(1-(float)cos(theta)) - 25*(float)sin(theta);

        }
        previous = _setcolor( T_WHITE | T_BRIGHT );

        line( (short)x[0], (short)y[0], (short)x[1], (short)y[1]);
        line( (short)x[2], (short)y[2], (short)x[3], (short)y[3]);
        line( (short)x[4], (short)y[4], (short)x[5], (short)y[5]);

        _setcolor( previous );
}
/*---------------------------------
  Function:     Init_ac_orientations()

  File:         t1object.c

  Parameters:   None

  Returned:     None

  Description:  Draws the aircraft in the eight possible orientations,
                saving each image in a buffer.  Assigns the global
                aircraft pointers to the starting locations of each
                buffer for future drawing of any aircraft.

 ----------------------------------
*/

void Init_ac_orientations( void )
{
        unsigned image_size;
        char *image;
        int i;

        /*
         Set active page to non visual page
        */
        _setactivepage( 1 );
```

```
        /* determine image size of each aircraft drawing */
        image_size = _imagesize( device_x( 0 ), device_y( 0 ),
                                                        device_x( 50 ), device_y( 50 ) ),

        /* draw and save each of the eight orientations */
        for( i = 0, i <8, i++)
        {
                /* clear area where image will be drawn */
                custom_bar( 0, 0, 50, 50, T_BLACK ),

                /* draw image */
                Draw_plane((float)(i*45)),

                /* allocate memory */
                aircraft_ptr[i] = (char*)malloc( image_size ),
                /* place image into memory */
                _getimage( device_x( 0 ), device_y( 0 ), device_x( 50), device_y( 50 ),
                                        aircraft_ptr[i]),
        }
        _clearscreen(_GCLEARSCREEN),
        /*
          Set active page back to visual page
        */
        _setvisualpage( 1 ),

}
/* ---------------------------------
  Function      Free_aircraft()

  File.         t1object c

  Parameters    None

  Returned      None

  Description   Frees the memory buffers holding the aircraft
                                        images in various orientations
-----------------------------------
*/
void Free_ac( void )
{
        int i,
        for( i=0, i<8, i++ )
                free(aircraft_ptr[i]),
}

/* ---------------------------------
  Function      Draw_aircraft_problem()

  File          t1object c

  Parameters    orientation of aircraft
                                        position of aircraft on screen

  Returned      None

  Description   draws the aircraft on screen at the position and
                                        and orientation specified

-----------------------------------
*/
void Draw_aircraft_problem( short ac_orientation, short ac_position )
{

  char *image,
  short x, y;

  /* determine aircraft orientation required */
  switch( ac_orientation )
```

```
  {
                        case 0:   image = aircraft_ptr[0]; break;

                        case 45:  image = aircraft_ptr[1]; break;

                        case 90:  image = aircraft_ptr[2]; break;

                        case 135: image = aircraft_ptr[3]; break;

                        case 180: image = aircraft_ptr[4]; break;

                        case 225: image = aircraft_ptr[5]; break;

                        case 270: image = aircraft_ptr[6]; break;

                        case 315: image = aircraft_ptr[7]; break;
  }

  /* determine aircraft position required relative to center of screen.
          North (0 deg bearing) being up on the screen
  */
  switch( ac_position )
  {
                        case 0:    x = 400; y = 550; break;

                        case 45:   x = 575; y = 475; break;

                        case 90:   x = 650; y = 300; break;

                        case 135:  x = 575; y = 125; break;

                        case 180:  x = 400; y = 50;  break;

                        case 225:  x = 225; y = 125; break;

                        case 270:  x = 150; y = 300; break;

                        case 315:  x = 225; y = 475; break;

  }

  /* place aircraft image on screen */
  _putimage( device_x( x-25 ), device_y( y+25 ), image, _GPSET );

}
/*--------------------------------
  Function:     Draw_example_aircraft_problem()

  File:         t1object.c

  Parameters:   orientation of aircraft.
                                        position of aircraft on screen.

  Returned:     None

  Description:  draws the aircraft on screen at the position and
                                        and orientation specified.

--------------------------------------
*/
void Draw_example_aircraft_problem( short ac_orientation, short ac_position )
{

  char *image;
  short x, y;

  /* determine aircraft orientation required */
  switch( ac_orientation )
  {
```

```
                    case 0   image = aircraft_ptr[0], break,

                    case 45  image = aircraft_ptr[1], break,

                    case 90  image = aircraft_ptr[2], break,

                    case 135 image = aircraft_ptr[3], break,

                    case 180 image = aircraft_ptr[4], break,

                    case 225 image = aircraft_ptr[5], break,

                    case 270 image = aircraft_ptr[6], break,

                    case 315 image = aircraft_ptr[7], break,
  }

  /* determine aircraft position required relative to center of screen
          North (0 deg bearing) being up on the screen
  */
  switch( ac_position )
  {
                    case 0   x = 400, y = 575, break,

                    case 45  x = 541, y = 516, break,

                    case 90  x = 600, y = 375, break,

                    case 135 x = 541, y = 234, break,

                    case 180 x = 400, y = 175, break,

                    case 225 x = 259, y = 234, break,

                    case 270 x = 200, y = 375, break,

                    case 315 x = 259, y = 516, break,

  }

  /* place aircraft image on screen */
  _putimage( device_x( x-25 ), device_y( y+25 ), image, _GPSET ),

}

/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/


/* ---------------------------------
          Function     blue_bar(),
          File         TEST_1 C

          Parameters   None

          Returned     None

          Variables    None

          Description  makes the entire screen blue
          ---------------------------------
*/
void blue_bar( void )
{
          short previous,

          previous = _setcolor( T_BLUE ),
```

```
        _rectangle( _GFILLINTERIOR, device_x( 0 ), device_y( 595 ),
                                        device_x( 800 ), device_y( 0 ) );

        _setcolor( previous );
}


/* ---------------------------------
        Function:    up_black_bar();
        File:        TEST_1.C

        Parameters:   None

        Returned:     None

        Variables:    None

        Description:  draws black bar at top of screen
        ---------------------------------
*/
void up_black_bar( void )
{
        short previous;

        previous = _setcolor( T_BLACK );

        _rectangle( _GFILLINTERIOR, device_x( 0 ), device_y( 405 ),
                                        device_x( 800 ), device_y( 595 ) );

        _setcolor( previous );
}


/* ---------------------------------
        Function:    text_bar();
        File:        TEST_1.C

        Parameters:   None

        Returned:     None

        Variables:    None

        Description:  draws text bar
        ---------------------------------
*/
void text_bar( void )
{
        short previous;

        previous = _setcolor( T_BLUE );


         _rectangle( _GFILLINTERIOR, device_x( 0 ), device_y( 595 ),
                                                device_x( 800 ), device_y( 440 ) );

        _setcolor( previous );
}


/* ---------------------------------
        Function:    mid_text_bar();
        File:        TEST_1.C

        Parameters:   None

        Returned:     None

        Variables:    None
```

```
            Description    draws text bar
            --------------------------------
*/
void mid_text_bar( void )
{
            short previous,

            previous = _setcolor( T_BLUE ),

            _rectangle( _GFILLINTERIOR, device_x( 0 ), device_y( 425 ),
                                                    device_x( 800 ), device_y( 260 ) ),

            _setcolor( previous ),
}

/* --------------------------------
            Function      custom_bar(),
            File          TEST_1 C

            Parameters    None

            Returned      None

            Variables     None

            Description   draws a customized bar given coordinates of
                                corners of the bar
            --------------------------------
*/
void custom_bar( int x1, int y1, int x2, int y2, int color )
{
            short previous,

            previous = _setcolor( color ),

            _rectangle( _GFILLINTERIOR, device_x( x1 ), device_y( y1 ),
                                                    device_x( x2 ), device_y( y2 ) ),

            _setcolor( previous ),
}

/* --------------------------------
            Function      down_text_bar(),
            File          TEST_1 C

            Parameters    None

            Returned      None

            Variables     None

            Description   draws text bar
            --------------------------------
*/
void down_text_bar( void )
{
            short previous,

            previous = _setcolor( T_BLUE ),

            _rectangle( _GFILLINTERIOR, device_x( 0 ), device_y( 120 ),
                                                device_x( 800 ), device_y( 0 ) ),

            _setcolor( previous ),
}

/* --------------------------------
            Function      press_key(),
```

```
            File        TEST_1 C

            Parameters    None

            Returned      None

            Variables     None

            Description   draws a brown text bar and displays the
                                                'press any key to continue' message
            ---------------------------------
*/
void press_key( void )
{
            short previous,

            static unsigned char list[20],

            /*
                        The names of the fonts that are available on disk
            */
            static unsigned char *face[4] =
            {
                        "t'courier'",
                        "t'helv'",
                        "t'tms rmn'",
                        "t'modern'"
            },

            char *temp,

            unsigned image_size,


            /*
                        Copy previous background to memory
            */

            /* determine size of image ( bytes ) */
            image_size = _imagesize( device_x( 500 ), device_y( 125 ),
                                    device_x( 760 ), device_y( 160 ) ),

            /* allocate memory */
            temp = malloc( image_size ),

            /* place image into memory */
            _getimage( device_x( 500 ), device_y( 125 ),
                                    device_x( 760 ), device_y( 160 ), temp ),


            /* set the font for the press any key box */
            strcpy( list, face[2] ),
            strcat( list, "h15w12b" ),

            /* set the font */
            _setfont( list ),

            /* delay two seconds before drawing */
            wait_ticks( 36 ),

            /* first flush the keyboard buffer */
            while (kbhit())
                        getch(),

            previous = _setcolor( T_BROWN ),

            _rectangle( _GFILLINTERIOR, device_x( 500 ), device_y( 125 ),
                                    device_x( 760 ), device_y( 160 ) ),
```

```
        _moveto( device_x( 510 ), device_y( 155 ) );

        _setcolor( T_WHITE | T_BRIGHT );

        _outgtext("Press any key to continue");

        getch();

        _setcolor( previous );

        /* replace image on the screen */
        _putimage( device_x( 500 ), device_y( 160 ), temp, _GPSET );

        /* free up allocated memory */
        free( temp );


}

/* ---------------------------------
        Function:     example_sound_prompt();

        File:         TEST_1.C

        Parameters:   None

        Returned:     None

        Variables:    None

        Description:  prompts user to press any key to hear example
                                                warning time sound.
        ---------------------------------
*/
void example_sound_prompt( void )
{
        short previous;

        static unsigned char list[20];

        /*
                The names of the fonts that are available on disk
        */
        static unsigned char *face[4] =
        {
                "t'courier'",
                "t'helv'",
                "t'tms rmn'",
                "t'modern'"
        };

        char *temp;

        unsigned image_size;


        /*
                Copy previous background to memory
        */

        /* determine size of image ( bytes ) */
        image_size = _imagesize( device_x( 140 ), device_y( 120 ),
                                        device_x( 552 ), device_y( 155 ) );

        /* allocate memory */
        temp = malloc( image_size );

        /* place image into memory */
        _getimage( device_x( 140 ), device_y( 120 ),
```

```
                                        device_x( 552 ), device_y( 155 ), temp ),

        /* set the font for the press any key box */
        strcpy( list, face[2] ),
        strcat( list, "h15w12b" ),

        /* set the font */
        _setfont( list ),

        /* delay two seconds before drawing */
        wait_ticks( 36 ),

        /* first flush the keyboard buffer */
        while (kbhit())
                getch(),

        previous = _setcolor( T_BROWN ),

        _rectangle( _GFILLINTERIOR, device_x( 140 ), device_y( 120 ),
                                                device_x( 552 ), device_y( 155 ) ),

        _moveto( device_x( 150 ), device_y( 150 ) ),

        _setcolor( T_WHITE | T_BRIGHT ),

        _outgtext("To hear sound and continue press any key"),

        getch(),

        _setcolor( previous ),

        /* replace image on the screen */
        _putimage( device_x( 140 ), device_y( 155 )  temp, _GPSET ),

        /* free up allocated memory */
        free( temp ),
}
/* ---------------------------------
        Function        timeout_message(),

        File            TEST_1 C

        Parameters      None

        Returned        None

        Variables       None

        Description     A text bar displaying a message indicating
                                                timeout has occured and a new problem is being
                                                presented is flashed on screen for a brief moment
        ---------------------------------
*/
void timeout_message( void )
{
        short previous,

        static unsigned char list[20]

        /*
                The names of the fonts that are available on disk
        */
        static unsigned char *face[4] =
        {
                "t'courier'",
                "t'helv'",
                "t'tms rmn'",
                "t'modern'"
```

```
        };


        /* set the font for the press any key box */
        strcpy( list, face[2] );
        strcat( list, "h15w12b" );

        /* set the font */
        _setfont( list );

        previous = _setcolor( T_RED );

        _rectangle( _GFILLINTERIOR, device_x( 520 ), device_y( 245 ),
                                                device_x( 750 ), device_y( 180 ) );

        _setcolor( T_WHITE | T_BRIGHT );

        _moveto( device_x( 530 ), device_y( 240 ) );
        _outgtext("Time has elapsed!");
        _moveto( device_x( 530 ), device_y( 210 ) );
        _outgtext("This is a NEW pattern.");

        /* wait one second for user to read message flash */
        wait_ticks(16);

        /* clear message */
        _setcolor( T_BLACK );
        _rectangle( _GFILLINTERIOR, device_x( 520 ), device_y( 245 ),
                                                device_x( 750 ), device_y( 180 ) );

        _setcolor( previous );

}
/* ---------------------------------
        Function:     print_countdown();
        File:         TEST_1.C

        Parameters:    None

        Returned:      None

        Variables:     None

        Description:   prints count down message on the screen
        ---------------------------------
*/
void print_countdown( void )
{
        short previous;

        static unsigned char list[20];

        /*
                The names of the fonts that are available on disk
        */
        static unsigned char *face[4] =
        {
                "t'courier'",
                "t'helv'",
                "t'tms rmn'",
                "t'modern'"
        };

        char *temp, digit[3];
        int  counter;

        unsigned image_size;

        /* clear the screen */
```

```
_clearscreen(_GCLEARSCREEN);

/* set the font for the press any key box */
strcpy( list, face[2] );
strcat( list, "h15w12b" );

/* set the font */
_setfont( list );

previous = _setcolor( T_BROWN );
_rectangle( _GFILLINTERIOR, device_x( 60 ), device_y( 425 ),
                                        device_x( 740 ), device_y( 470 ) );

_setcolor( T_BLUE );
_rectangle( _GFILLINTERIOR, device_x( 230 ), device_y( 250 ),
                                        device_x( 570 ), device_y( 400 ) );

_moveto( device_x( 80 ), device_y( 460 ) );
_setcolor( T_WHITE | T_BRIGHT );
_outgtext("RESPOND AS QUICKLY AND AS ACCURATELY AS YOU CAN");

_moveto( device_x( 250 ), device_y( 390 ) );
_setcolor( T_WHITE | T_BRIGHT );
_outgtext("THE TEST WILL BEGIN IN");

_moveto( device_x( 340 ), device_y( 290 ) );
_setcolor( T_WHITE | T_BRIGHT );
_outgtext("SECONDS");

/* set the font for the press any key box */
strcpy( list, face[2] );
strcat( list, "h20w15b" );

/* set the font */
_setfont( list );

/* countdown from 10 to 1 */
digit[2] = '\0';
for (counter = 10; counter >= 1; counter--) {

        /* form digit string to be displayed on screen 9...8.. etc */
        if (counter >= 10) {
                digit[0] = '1'; digit[1] = '0';
                }
        else {
                digit[0] = ' '; digit[1] = counter + 48;
                }

        _setcolor( T_WHITE | T_BRIGHT );
        _moveto( device_x( 380 ), device_y( 340 ) );
        _outgtext(digit);

        /* delay for one second */
        wait_ticks( 18 );

        _setcolor( T_BLUE );
        _moveto( device_x( 380 ), device_y( 340 ) );
        _outgtext(digit);

}

_setcolor( previous );

/* set the font for the press any key box */
strcpy( list, face[2] );
strcat( list, "h15w12b" );

/* set the font */
_setfont( list );
```

```
        /* clear the screen */
        _clearscreen(_GCLEARSCREEN);
}
/* ---------------------------------
        Function:    full_text_bar();
        File:        TEST_1.C

        Parameters:   None

        Returned:     None

        Variables:    None

        Description:  draws text bar
        ---------------------------------
*/
void full_text_bar( void )
{
        short previous;

        previous = _setcolor( T_BLUE );

        _rectangle( _GFILLINTERIOR, device_x( 0 ), device_y( 550 ),
                                                device_x( 800 ), device_y( 35 ) );

        _setcolor( previous );
}
/* ---------------------------------
        Function:    full_black_bar();
        File:        TEST_1.C

        Parameters:   None

        Returned:     None

        Variables:    None

        Description:  draws black bar
        ---------------------------------
*/
void full_black_bar( void )
{
        short previous;

        previous = _setcolor( T_BLACK );

        ( _GFILLINTERIOR, device_x( 0 ), device_y( 600 ),
                                                device_x( 800 ), device_y( 0 ) );

        _setcolor( previous );
}

/* ---------------------------------
        Function:    display_test_name();
        File:        T1OBJECT_1.C

        Parameters:   None

        Returned:     None

        Variables:    None

        Description:  displays the name of a test for 2 seconds on
                        screen
        ---------------------------------
*/
void display_test_name( char *test_name )
{
```

```
                "t'helv'",
                "t'tms rmn'",
                "t'modern'"
        };

        /* Display digit centered at the top of the screen */
        strcpy( list, face[2] );
        strcat( list, "h40w32b" );

        /* set the font */
        _setfont( list );

        /* set text color to green */
        previous = _setcolor( T_RED );

        /* drawing brown rectangle */
        _rectangle( _GFILLINTERIOR, device_x( 225 ), device_y( 420 ),
                                        device_x( 575 ), device_y( 500 ) );
        /* reset drawing color */
        _setcolor( previous );

        /* Draw text on screen */
        _moveto( device_x( 330 ), device_y( 475 ) );

        /* output character */
        _outgtext( "BEGIN!" );

        /* wait one and a half seconds */
        wait_ticks( 27 );
}
/* ---------------------------------
        Function:     next_trial_message();
        File:         T1OBJECT_1.C

        Parameters:    None

        Returned:      None

        Variables:     None

        Description:   draws 'next trial message' on the screen
        ---------------------------------
*/
void next_trial_message( void )
{
        short previous;

        static unsigned char list[20];

        /*
                The names of the fonts that are available on disk
        */
        static unsigned char *face[4] =
        {
                "t'courier'",
                "t'helv'",
                "t'tms rmn'",
                "t'modern'"
        };

        /* Display digit centered at the top of the screen */
        strcpy( list, face[2] );
        strcat( list, "h40w32b" );

        /* set the font */
        _setfont( list );

        /* set text color to green */
        previous = _setcolor( T_BROWN );
```

```
        /* drawing brown rectangle */
        _rectangle( _GFILLINTERIOR, device_x( 10 ), device_y( 420 ),
                                                                    device_x( 790 ),
device_y( 500 ) );
        /* reset drawing color */
        _setcolor( previous );

        /* Draw text on screen */
        _moveto( device_x( 30 ), device_y( 475 ) );

        /* Make error sound */
        warble( 5 );

        /* output character */
        _outgtext( "PRESS ANY KEY TO START NEXT TRIAL" );

        /* get user key press */
        getch();
}
/* ---------------------------------
        Function:     next_instruction_message();
        File:         T1OBJECT_1.C

        Parameters:    None

        Returned:      None

        Variables:     None

        Description:   draws 'next instruction message' on the screen
        ---------------------------------
*/
void next_instruction_message( void )
{
        short previous;

        static unsigned char list[20];

        /*
                The names of the fonts that are available on disk
        */
        static unsigned char *face[4] =
        {
                "t'courier'",
                "t'helv'",
                "t'tms rmn'",
                "t'modern'"
        };

        /* Display digit centered at the top of the screen */
        strcpy( list, face[2] );
        strcat( list, "h40w32b" );

        /* set the font */
        _setfont( list );

        /* set text color to green */
        previous = _setcolor( T_BROWN );

        /* drawing brown rectangle */
        _rectangle( _GFILLINTERIOR, device_x( 10 ), device_y( 420 ),
                                                                    device_x( 790 ),
device_y( 500 ) );
        /* reset drawing color */
        _setcolor( previous );

        /* Draw text on screen */
        _moveto( device_x( 30 ), device_y( 475 ) );
```

```
/* ---------------------------------------------------
        Name        TEST_1 C
        Type        Routines to implement the first
                                                student test
                                                Mental Rotation Program
        Language    Microsoft QuickC version 2
        ---------------------------------------------------
*/

#include <graph h>
#include <conio h>
#include <malloc h>
#include <stdio h>
#include "typ_init h"
#include "video h"
#include "t_colors h"
#include "stats h"
#include "t1object h"

#include "menu h"
#include "sound h"
#include "getkey h"
#include "box h"

#define AC_ORIENTATION 0
#define AC_POSITION 1

#define NUM_TRIALS  2
#define NUM_PROBLEMS 64  /* Note  this parameter is also defined in t1object c */

#define NFONTS 4
#define GRID_CHAR_FONT  2

#define FRAME_0_0     "          Mental Rotation/Orientation Test"

#define FRAME_1_1     "The object of this test is to measure the time and accuracy it"
#define FRAME_1_2     "takes for you to orient where the triangle in the center of the"
#define FRAME_1_3     "screen is in relation to the nose of the aircraft icon "
/*#define FRAME 1_2     show the triangle on the left side & aircraft on left
*/
/*#define FRAME 1_3     "Press Enter to See example"
*/
#define FRAME_2_1     "As you can see, the aircraft will appear at one of the eight"
#define FRAME_2_2     "45 degree points around the center triangle "
#define FRAME_3_1     "You are to respond by using the numeric keypad on the right side"
#define FRAME_3_2     "of the keyboard   The eight outside numeric keys correspond to"
#define FRAME_3_3     "the location of the triangle relative to the aircraft's nose "
#define FRAME_4_1     "The numeric keys correspond to the angles as follows "
#define FRAME_4_2     "#8 = 0,       #9 = 045,      #6 = 090,      #3 = 135,"
#define FRAME_4_3     "#2 = 180,     #1 = 225,      #4 = 270,      #7 = 315 "
#define FRAME_5_1     "For this example, the triangle is at the 90 degree point from"
#define FRAME_5_2     "the nose of the aircraft   Therefore, the correct answer would"
#define FRAME_5_3     "be to press the #6 key "
#define FRAME_6_1     "Let's try another example   "
#define FRAME_7_1     "Where is the location of the triangle in relation to the nose"
#define FRAME_7_2     "of the aircraft?"
#define FRAME_7_3     "Press the key which corresponds to the correct orientation "
#define FRAME_8_1     "Let's try one last example  Where is the location of the"
#define FRAME_8_2     "triangle in relation to the nose of the aircraft?"
#define FRAME_8_3     "Press the key which corresponds to the correct orientation "
#define FRAME_9_1     "Your score will be based on speed as well as accuracy "
#define FRAME_9_2     "Therefore, please try to respond as quickly as possible,"
#define FRAME_9_3     "BUT also as accurately as you can!"
#define FRAME_9_4     "After you select your answer to each trial, the next trial'
#define FRAME_9_5     "will immediately begin on the next screen "
#define FRAME_9_6     "There are 64 problems in the test "
#define FRAME_9_7     "There will be a short break in the middle of the test "

/*
```

```
            Variables to keep track of reaction time
            and answer for each test
*/
            TEMP results[NUM_PROBLEMS],


/* Error message data */
char *error_box_4_01[] =
            {
            " Error Message #4 01 ",
            "",
            " You failed to score 55% or greater ",
            " on the preview test  ",
            "",
            "< Press any key >",
            NULL
},

char *error_box_5_01[] =
  {
  " Error Message #5 01 ",
  "",
  " Unable to register fonts for this test ",
  "",
  " The following files must be in the ",
  " the current directory for this test ",
  " to run  ",
  "",
  "    1) HELVB FON ",
  "    2) COURB FON ",
  "    3) TMSRB FON ",
  "",
  "< Press any key >",
  NULL
  },

char *info_box_19[] =
            {
            " Test #1 ",
            "",
            "       TEST COMPLETE ",
            "",
            "< Press any key to continue >",
            NULL
            },

/* routine to free memory held by buffers with aircfaft drawn in various
   orientations
*/
/* ---------------------------------
            Function      Display_test1_instructions
            File          TEST_1 C

            Parameters     None

            Returned       None

            Variables      None

            Description    Displays instructions for test#1
            ---------------------------------
*/

void Display_test1_instructions( void )
{
            static unsigned char list[20],

            int response,
```

```
/*
        The names of the fonts that are available on disk
*/
static unsigned char *face[NFONTS] =
{
        "t'courier'",
        "t'helv'",
        "t'tms rmn'",
        "t'modern'"
};

/* flush all buffers */
flushall();

/*
 * frame #0  (Title of Test)
 */

/* set font type and size */
strcpy( list, face[2] );
strcat( list, "h18w14b" );

/* set the font */
_setfont( list );

display_test_name( FRAME_0_0 );

/*
 * frame #1
 */

/* set font type and size */
strcpy( list, face[2] );
strcat( list, "h18w14b" );

/* set the font */
_setfont( list );


Draw_example_background();
Draw_example_aircraft_problem( 0, 270 );


/* create text bar */
down_text_bar();

/* display text for frame 1 */
_moveto( device_x(10), device_y(120) );
_outgtext(FRAME_1_1);
_moveto( device_x(10), device_y(80) );
_outgtext(FRAME_1_2);
_moveto( device_x(10), device_y(40) );
_outgtext(FRAME_1_3);


/* wait for key press */
press_key();

/*
 * frame #2
 */

/* set the font */
_setfont( list );

/* refresh text bar */
down_text_bar();

/* display text for frame 2 */
```

```
    _moveto( device_x(10), device_y(120) ),
    _outgtext(FRAME_2_1),
    _moveto( device_x(10), device_y(80) ),
    _outgtext(FRAME_2_2),

    /* set font type and size  back to normal */
    strcpy( list, face[2] ),
    strcat( list, "h18w14b" ),

    /* set the font */
    _setfont( list ),

    /* wait for key press */
    press_key(),

    /*
     * frame #3
     */


    /* set the font */
    _setfont( list ),

    /* create text bar for text */
    down_text_bar(),

    /* display text */
    _moveto( device_x(10), device_y(120) ),
    _outgtext(FRAME_3_1),
    _moveto( device_x(10), device_y(80) ),
    _outgtext(FRAME_3_2),
    _moveto( device_x(10), device_y(40) ),
    _outgtext(FRAME_3_3),


    /* wait for key press */
    press_key(),

    /*
     * frame #4
     */

    /* set the font */
    _setfont( list ),

    /* create text bar */
    down_text_bar(),

    /* display text for frame 4 */
    _moveto( device_x(10), device_y(120) ),
    _outgtext(FRAME_4_1),
    _moveto( device_x(10), device_y(80) ),
    _outgtext(FRAME_4_2),
    _moveto( device_x(10), device_y(40) ),
    _outgtext(FRAME_4_3),

    /* wait for user to press key */
    press_key(),

    /*
     * frame #5
     */

    /* set the font */
    _setfont( list ),

    /* create text bar */
    down_text_bar(),
```

```
/* display text for frame 5 */
_moveto( device_x(10), device_y(120) );
_outgtext(FRAME_5_1);
_moveto( device_x(10), device_y(80) );
_outgtext(FRAME_5_2);
_moveto( device_x(10), device_y(40) );
_outgtext(FRAME_5_3);

/* wait for user to press key */
press_key();

/*
 * frame #6
 */

/* create text bar */
down_text_bar();

/* set the font */
_setfont( list );

/* display text for frame 6 */
_moveto( device_x(10), device_y(120) );
_outgtext(FRAME_6_1);

/* wait for user to press key */
press_key();

/*
 * frame #7
 */

/* set the font */
_setfont( list );

/* erase old example from screen */
_clearscreen( _GCLEARSCREEN );


Draw_example_background();
Draw_example_aircraft_problem( 90, 45 );

/* create text bar */
down_text_bar();

/* display text for frame 7 */
_moveto( device_x(10), device_y(120) );
_outgtext(FRAME_7_1);
_moveto( device_x(10), device_y(80) );
_outgtext(FRAME_7_2);
_moveto( device_x(10), device_y(40) );
_outgtext(FRAME_7_3);

/* get response from user and check
then display appropriate message */

response = getch();

if ( response == '3' )
 {
  _moveto( device_x(10), device_y(200) );
  _outgtext( "That's correct!" );
 }
else
 {
  _moveto( device_x(10), device_y(400) );
  _outgtext( "Sorry, the triangle is in the" );
  _moveto( device_x(10), device_y(360) );
  _outgtext( "135 degree position (#3 key)." );
```

```
 }

/* wait for user to press key */
press_key();

/*
 * frame #8
 */

/* set the font */
_setfont( list );

/* erase old example from screen */
_clearscreen( _GCLEARSCREEN );


Draw_example_background();
Draw_example_aircraft_problem(135,180 );

/* create text bar */
down_text_bar();

/* display text for frame 7 */
_moveto( device_x(10), device_y(120) );
_outgtext(FRAME_8_1);
_moveto( device_x(10), device_y(80) );
_outgtext(FRAME_8_2);
_moveto( device_x(10), device_y(40) );
_outgtext(FRAME_8_3);

/* get response from user and check
then display appropriate message */

response = getch();

if ( response == '1' )
 {
  _moveto( device_x(10), device_y(200) );
  _outgtext( "That's correct!" );
 }
else
 {
  _moveto( device_x(10), device_y(400) );
  _outgtext( "Sorry, the triangle is in the" );
  _moveto( device_x(10), device_y(360) );
  _outgtext( "225 degree position (#1 key)." );

 }


/* wait for user to press key */
press_key();

/*
 * frame #9
 */


/* set font type and size  back to normal */
strcpy( list, face[2] );
strcat( list, "h18w14b" );

_clearscreen( _GCLEARSCREEN );
_setfont( list );

/* set blue background */
blue_bar();
```

```
        /* display text for frame 9 */
        _moveto( device_x(10), device_y(510) ),
        _outgtext(FRAME_9_1),
        _moveto( device_x(10), device_y(470) ),
        _outgtext(FRAME_9_2),
        _moveto( device_x(10), device_y(430) ),
        _outgtext(FRAME_9_3),
        _moveto( device_x(10), device_y(390) ),
        _outgtext(FRAME_9_4),
        _moveto( device_x(10), device_y(350) ),
        _outgtext(FRAME_9_5),
        _moveto( device_x(10), device_y(310) ),
        _outgtext(FRAME_9_6),
        _moveto( device_x(10), device_y(270) ),
        _outgtext(FRAME_9_7),

        /* wait for user to press key */
        press_key(),

        /* print countdown message on the screen */
        print_countdown(),

        /* print begin message on the screen */
        begin_message(),

        /* clear the screen */
        _clearscreen( _GCLEARSCREEN ),
}


/* ---------------------------------
        Function      Test_1(),
        File          TEST_1 C

        Parameters    None

        Returned      None
        0
        Variables     None

        Description   Procedure to execute test_1  This test determines
                      the mental rotation capabilities of a person
                      An airplane icon is presented in any of eight orientations
                      (0,45,135,180,225,270,315) on the screen  The icon itself
                      is placed in one of eight positions on the screen  The
                      object is to determine the angular position of the center
                      of the screen if the user were in the airplane and facing
                      forward
        ---------------------------------
*/


void test_1( STUDENT_RECORD *new_student )
{

        int  num_trials, num_statements,
        char key_field[9],

        short previous,
        double t,
        int n,
        int test = 1,
        int *save_error_box,
        int *save_info_box,
        long image_size,
```

```
FILE *debug_data;

/* array to hold questions.  Format of questions is:
        question[n][0] = aircraft_orientation (deg)
        question[n][1] = aircraft_position   (deg)
*/

short question[NUM_PROBLEMS][2] = {
                                    { 0,  0}, { 45,225}, {135, 90}, {180,315}, {270,180},
                                    { 0, 90}, { 45,315}, {135,180}, {225, 45}, {270,270},
                                    { 90,  0}, {135,225}, {225, 90}, {270,315}, {315,315},
                                    { 0,225}, { 90, 90}, {270,225}, {225,225}, {315, 90},
                                    {135,315}, {225,180}, {315, 45}, { 0,270}, { 90,135},
                                    { 45,  0}, { 90,225}, {180, 90}, {225,315}, {315,180},
                                    { 45, 45}, { 90,270}, {180,135}, {270,  0}, {315,225},
                                    { 45,135}, {135,  0}, {180,225}, { 0, 45}, {180,  0},
                                    { 45, 90}, { 90,315}, {180,180}, {270, 45}, {315,270},
                                    { 0,315}, {270,135}, {180, 45}, { 90, 45}, {315,135},
                                    {270, 90}, { 0,135}, { 45,270}, {135,135}, { 45,180},
                                    { 0,180}, {225,270}, {135,270}, {225,135}, {315,  0},
                                    {225,  0}, {135, 45}, {180,270}, { 90,180}};

/* array that holds correct answer key press for all questions */
short answer[64] = { '2', '8', '3', '7', '6', '4', '6', '1', '8', '2',
                     '6', '4', '9', '1', '2', '9', '2', '3', '2', '7',
                     '8', '3', '4', '6', '1', '3', '7', '6', '4', '9',
                     '2', '8', '3', '4', '6', '4', '9', '1', '1', '8',
                     '1', '9', '2', '7', '3', '3', '9', '9', '3', '8',
                     '8', '7', '9', '2', '7', '8', '1', '7', '6', '1',
                     '7', '6', '4', '4'
                   };

static unsigned char list[20];

/*
        The names of the fonts that are available on disk
*/
static unsigned char *face[NFONTS] =
{
        "t'courier'",
        "t'helv'",
        "t'tms rmn'",
        "t'modern'"
};

key_field[0] = '1'; key_field[1] = '2'; key_field[2] = '3';
key_field[3] = '4'; key_field[4] = '6'; key_field[5] = '7';
key_field[6] = '8'; key_field[7] = '9'; key_field[8] = '*';

debug_data =  fopen("debug.fil","w+");

/*
        Read header from all font files
        in current directory
*/
if ( _registerfonts( "*.fon" ) < 0 )
{
        /*
                set error box color to red
                set error text color to white
        */
        menu_back_color( BK_RED );
        menu_text_color( T_WHITE | T_BRIGHT );

        /* Display error_box_5_01 */
        save_error_box = menu_message( 6, 8, error_box_5_01 );

        /* Make error sound */
        warble( 5 );
```

```
                /* Get keypress from user */
                getch(),

                /* Erase error_box_5_01 */
                menu_erase( save_error_box ),

                /*
                        set box color back to cyan
                        set text color back to black
                */
                menu_back_color( BK_WHITE ),
                menu_text_color( T_BLACK ),
        }
        else
        {
                /* Place graphics adapter into videomode */
                best_graph_mode(),

                /* Display digit centered at the top of the screen */
                strcpy( list, face[GRID_CHAR_FONT] ),
                strcat( list, "h40w32b" ),

                /* set the font */
                _setfont( list ),

                /* set text color to blue -- same as background */
                previous = _setcolor( T_BROWN ),

                /* reset drawing color */
                _setcolor( previous ),

                /* Initialize pointers to buffers that draw the aircraft in any
                   of the eight given orientations */
                Init_ac_orientations(),

                /* Display test 1 instructions */
                Display_test1_instructions(),

                /* run test in two trials */
                for ( num_trials = 0, num_trials < NUM_TRIALS  , num_trials++ )
                {
                        for ( num_statements = num_trials*NUM_PROBLEMS/2,
                                num_statements < NUM_PROBLEMS/2+num_trials*NUM_PROBLEMS/2,
                                num_statements++ )
                        {
                                /* flush the keyboard buffer */
                                while (kbhit())
                                        getch(),

                                /* clear screen and display background */
                                _clearscreen (_GCLEARSCREEN ),
                                Draw_background(),

                                /* check for timeout and display appropriate message */
                                /* NOTE  test is initialized to 1 to ensure timeout message is not
                                                erroneously displayed for first problem */
                                if ( test == 0 )
                                        timeout_message()

                                Draw_aircraft_problem( question[num_statements][AC_ORIENTATION],
question[num_statements][AC_POSITION] ),

                                /* set timer with 2 minute timeout and 10 second warning 'beep' feature */
                                results[num_statements] reaction_time =  student_timer( &test, key_field,
(unsigned)120, (unsigned)10),
                                results[num_statements] answer       = test,
```

```
                }

                /* return if this is a demonstration test */
                if ( new_student->test_no < 0 )
                {
                        /* Return to text mode */
                        text_mode(),

                        /* Set foreground and background colors for program */
                        _setbkcolor( BK_CYAN ),
                        _settextcolor( T_BLACK ),

                        /* Fill the background */
                        box_charfill( 1, 1, 25, 80, 178 ),

                        /* Return memory used by fonts */
                        _unregisterfonts(),

                        /* free up memory used by simple figures */
                        Free_ac(),

                        /* exit test */
                        return,
                }

                /* Statistical analysis of test results */
                stats_test_1( results, new_student, answer, num_trials ),

                if (num_trials < NUM_TRIALS - 1)
                {
                        /* clear the screen */
                        full_black_bar(),

                        /* ask user to press key to start the next trial */
                        next_trial_message(),

                        /* clear the screen */
                        full_black_bar(),

                        /* print countdown message on the screen */
                        print_countdown(),

                }
                else
                {
                        /* set graphics background to black */
                        _setbkcolor( _BLACK ),
                        _clearscreen( _GCLEARSCREEN ),

                        /* clear the screen */
                        full_black_bar(),

                        /* ask user to press key to end test */
                        _clearscreen( _GCLEARSCREEN ),
                        test_complete_message(),

                        /* clear the screen */
                        full_black_bar(),

                }

        }

}
/* Set text mode */
text_mode(),

/* Fill the background */
```

```
box_charfill( 1, 1, 25, 80, 178 );

/*
        set information box color to green
        set information box text color to white
*/
menu_back_color( BK_GREEN );
menu_text_color( T_WHITE | T_BRIGHT );

/* Display information_box_19 */
save_info_box = menu_message( 8, 8, info_box_19 );

getch();

/* Erase information_box_19 */
menu_erase( save_info_box );

/*
        set box color back to cyan
        set text color back to black
*/
menu_back_color( BK_WHITE );
menu_text_color( T_BLACK );

/* Free memory taken up by fonts */
_unregisterfonts();

/* free up memory used by simple figures */
Free_ac();

/* Set foreground & background colors for program */
_setbkcolor( BK_CYAN );
_settextcolor( T_BLACK );
```

```
/* ---------------------------------------------------
   Name:         TMANAGER.C
   Type:         Routines to execute student tests.
                 Air Traffic Control Screening Program
   Language:     Microsoft QuickC version 2
   ---------------------------------------------------
*/

#include <graph.h>
#include <conio.h>
#include <malloc.h>
#include "typ_init.h"
#include "video.h"
#include "test_1.h"


/* ---------------------------------
   Function:     Test_manager();
   File:         TMANAGER.C

   Parameters:   student       pointer to student record

   Returned:     None

   Variables:    None

   Description:  Executes

   ---------------------------------
*/

void Test_manager( STUDENT_RECORD *new_student )
{
   /* Start test #1 */
            test_1( new_student );

}
```

```
/* ---------------------------------------------------------
  Name        VIDEO C
  Type        Routines to implement virtual
                    display area for ATC graphic based
                    tests
                    Air Traffic Control Screening Program
  Language    Microsoft QuickC version 2

        Last Revision  06/16/92  Gordon Jones

        Note        Structure for _getvideoconfig() as visible to user

                    struct videoconfig {
                      short numxpixels,     number of pixels on X axis
                      short numypixels,     number of pixels on Y axis
                      short numtextcols,    number of text columns available
                      short numtextrows,    number of text rows available
                      short numcolors,      number of actual colors
                      short bitsperpixel,   number of bits per pixel
                      short numvideopages,  number of available video pages
                      short mode,           current video mode
                      short adapter,        active display adapter
                      short monitor;        active display monitor
                      short memory,         adapter video memory in K bytes
                    },

  ---------------------------------------------------------
*/

#include <graph h>
#include <stdio h>
#include <malloc h>
#include <conio h>
#include <stdlib h>
#include <time h>
#include "t_colors h"
#include "menu h"
#include "video h"
#include "mk_fp h"

#pragma pack(1)

#define VH 600                  /* height of virtual window     */
#define VW 800                  /* width of virtual window      */

static double x_trans = 0 0,        /* scaling factor for converting */
static double y_trans = 0 0,        /* from virtual to device coords */
static int max_y = 0,



/* Error message data */
char *error_box_2_01[] =
  {
  " Error Message #2 01 ",
  "",
  " UNABLE TO TURN ON GRAPHICS MODE ",
  "",
  " The Monochrome Display Adapter ",
  " installed in this computer ",
  " does NOT support graphics  ",
  " Graphics capability is needed  "
  "",
  "< Press any key >",
  NULL
  },


/*
```

```
  Procedure to place video adapter
  in text mode
*/

void text_mode( void )
{
  _setvideomode( _DEFAULTMODE ),
}


/*
  Procedure to place video adapter
  in best graphics mode
*/

void best_graph_mode( void )
{
  int *save_error_box,
  short best,
  struct videoconfig grconfig,

  /*
    place information about video
    adapter into structure variable
    grconfig
  */
  _getvideoconfig( &grconfig ),
   switch ( grconfig adapter ) {

    /* Monochrome Display Adapter */
    /* case _MDPA   best = -1,          break,

    /* Color Graphics Adapter */
    case _CGA    best = _MRES4COLOR,   break,

    /* Enhanced Graphics Adapter */
    case _EGA    best = _ERESCOLOR,    break,

    /* Video Graphics Array */
    case _VGA    best = _ERESCOLOR,    break,

    /* Multicolor Graphics Adapter */
    case _MCGA   best = _ERESCOLOR,    break,

    /* Hercules Graphics Card */
    case _HGC    best = _HERCMONO,     break,

  }
  if ( best != -1 ) {
    /*
          Set best video mode
    */
    _setvideomode( best ),

    /*
          Initialize video variables
    */
    x_trans = x_factor(),
    y_trans = y_factor(),
    max_y = maximum_y(),
    }
  else {
    /*
          Error - Monochrome Display Adapter
          cannot support graphics
    */

    /*
          set error box color to red
```

```
            set error text color to white
        */
        menu_back_color( BK_RED ),
        menu_text_color( T_WHITE | T_BRIGHT ),

        /* Display error_box_1 */
        save_error_box = menu_message( 10, 8, error_box_2_01 ),

        getch(),

        /* Erase error_box_1 */
        menu_erase( save_error_box ),

        /*
            set box color back to cyan
            set text color back to black
        */
        menu_back_color( BK_CYAN ),
        menu_text_color( T_BLACK ),
    }
}


/*
  Function to calculate scaling factor
  along the x axis
*/

double x_factor( void )
{
        /* max number of pixels - x axis  */
        int maxx,

  struct videoconfig video,

  /*
    place information about video
    adapter into structure variable
    video
  */
  _getvideoconfig( &video ),

  maxx = video numxpixels - 1,

  /* Calculate scaling factor for x axis */
  return( ( double ) ( maxx ) / VW ),
}


/*
  Function to calculate scaling factor
  along the y axis
*/

double y_factor( void )
{
  int maxy,                     /* max number of pixels - y axis */

  struct videoconfig video

  /*
    place information about video
    adapter into structure variable
    video
  */
  _getvideoconfig( &video ),

  maxy = video numypixels - 1,
```

```
  /* Calculate scaling factor for x axis */
  return( ( double ) ( maxy ) / VH ),
}


/*
  Function that returns maximum y
  coordinate for video adapter
*/

int maximum_y( void )
{
  struct videoconfig video,

  /*
    place information about video
    adapter into structure variable
    video
  */
  _getvideoconfig( &video ),

  return( video numypixels - 1 ),
}


/*
  Function to map virtual x coordinate
  to device x coordinate
*/

int device_x( register int virtual_x )
{
  return ( int ) ( x_trans * virtual_x ),
}


/*
  Function to map virtual y coordinate
  to device y coordinate
*/

int device_y( register int virtual_y )
{
  return ( int ) ( max_y - ( y_trans * virtual_y )),
}


/* ---------------------------------
  Function      line(),
  File          VIDEO C

  Parameters
    (input)     x1,y1           x and y coordinate of start of
                                        line
                x2,y2           x and y coordinate of end of
                                        line

  Returned      Nothing

  Variables     None

  Description   Draws a line using the virtual coordinate system
                implemented in this unit on the screen  The line
                is drawn from x1,y1 to x2,y2 in the current color

  ---------------------------------
*/

void line( int x1, int y1, int x2, int y2 )
```

```
{
  /* Move cursor position to start of line */
  _moveto( device_x( x1 ), device_y( y1 ) ),

  /* Draw line from x1,y1 to x2,y2 */
  _lineto( device_x( x2 ), device_y( y2 ) ),
}


/* ---------------------------------
  Function.     bresenham()

  Parameters
   (input)    x1     X-coordinate for first point
   (input)    y1     Y-coordinate for first point
   (input)    x2     X-coordinate for second point
   (input)    y2     Y-coordinate for second point

  Returned     Integer buffer containing points on line

  Variables    xi     X increment direction
                      yi     Y increment direction
                      dx     Relative change in x-coordinate
                      dy     Relative change in y-coordinate
                      xp     Current point along the line
                      yp     Current point along the line
                      cx     Accumulated x increments
                      cy     Accumulated y increments
                      buf    Pointer to returned buffer
                      ndx    Index into buf for each coordinate
                      i      Looping index

  Description  Builds a table of coordinates that form a line
                                                    connecting two given points

          Note         Bresenham function used because quicker than
                                                    standard Quick C fill function calls

                                                    For information on how this function works please
                                                    review graphics textbook
  ---------------------------------
*/

int *bresenham( int x1, int y1, int x2, int y2 )
{
   unsigned xi, yi, dx, dy, xp, yp, cx, cy,
   int *buf,
   int ndx = 1,
   int i,

   /* Right to left from first point to second? */
   if ( x2 < x1 )
          {
          dx = x1 - x2,
          xi = -1,
          }

   /* Must be left to right from first point to second */
   else
          {
          dx = x2 - x1,
          xi = 1,
          }

   /* Is first y-coordinate greater than second? */
   if ( y2 < y1 )
          {
          dy = y1 - y2,
          yi = -1,
```

```
        }

/* Second y-coordinate must be greater than first */
else
        {
        dy = y2 - y1;
        yi = 1;
        }

/* Set the working point to the first point */
xp = x1;
yp = y1;

/* Is the line more vertical than horizontal? */
if ( dx < dy )
        {

        /* Start with the accumulated count at halfway point */
        cy = dy >> 1;

        /* Allocate memory for the buffer */
buf = (int *)malloc( ((y2 - y1 + yi) * yi) * 4 + 2 );
        if ( buf == NULL )
           {
           printf( "Not enough memory for bresenham()\n" );
           exit( 1 );
           }

        /* Until we get to the last point */
        while ( yp != y2 )
           {

           /* Put the current point in the buffer */
           buf[ndx++] = xp;
           buf[ndx++] = yp;

           /* Accumulate the relative counts */
           cy += dx;
           yp += yi;

           /* Is it time to change x-coordinate? */
           if ( dy < cy )
                   {

                   /* Reset the accumulating count */
                   cy -= dy;

                   /* Change the X value */
                   xp += xi;
                   }
           }
        }

/* Line must be more horizontal than vertical */
else
        {

        /* Start with the accumulated count at halfway point */
        cx = dx >> 1;

        /* Allocate memory for the buffer */
buf = (int *)malloc( ((x2 - x1 + xi) * xi) * 4 + 2 );
        if ( buf == NULL )
           {
           printf( "Not enough memory for bresenham()\n" );
           exit( 1 );
           }

        /* Until we get to the last point */
```

```
        while ( xp != x2 )
          {

          /* Put the current point in the buffer */
          buf[ndx++] = xp;
          buf[ndx++] = yp;

          /* Accumulate the relative counts */
          cx += dy;
          xp += xi;

          /* Is it time to change y-coordinate? */
          if ( dx < cx )
                {

                /* Reset the accumulating count */
                cx -= dx;

                /* Change the Y value */
                yp += yi;
                }
          }
        }

        /* Save the last point in the buffer */
        buf[ndx++] = x2;
        buf[ndx++] = y2;

        /* Save the number of points at head of buffer */
        buf[0] = ndx >> 1;

        /* Return the buffer */
        return ( buf );
}


/* ---------------------------------
  Function      triangle()

  Parameters
    (input)     type    LINED (outline) or SOLID (filled)
    (input)     x1      X-coordinate at first point
    (input)     y1      Y-coordinate at first point
    (input)     x2      X-coordinate at second point
    (input)     y2      Y-coordinate at second point
    (input)     x3      X-coordinate at third point
    (input)     y3      Y-coordinate at third point

  Returned      (function returns nothing)

  Variables     buf12   Points along line from point 1 to 2
                buf23   Points along line from point 2 to 3
                buf13   Points along line from point 1 to 3
                xleft   Points along left side of triangle
                xright  Points along right side of triangle
                i       Looping index
                ymin    Minimum Y point of triangle
                ymax    Maximum Y point of triangle
                xmin    Minimum X point of triangle
                xmax    Maximum X point of triangle
                x       X-coordinates along triangle edges
                y       Y-coordinates along triangle edges
                numy    Number of Y-coordinates in triangle

        Description     Draws a triangle, optionally filled in

        Note            Bresenham function used because quicker than
                                                standard Quick C fill function calls
```

For information on how this function works please review graphics textbook.

```
        ---------------------------------
*/

void triangle( int type, int x1, int y1, int x2, int y2, int x3, int y3 )
{
   int *buf12, *buf23, *buf13;
   int *xleft, *xright;
   int i, ymin, ymax, xmin, xmax;
   int x, y, numy;

   if ( type == LINED )

          /* Draw only the outline */
          {
          _moveto( x1, y1 );
          _lineto( x2, y2 );
          _lineto( x3, y3 );
          _lineto( x1, y1 );
          }

   else

          /* Fill in solid area */
          {

          /* Determine minimum and maximum y-coordinates */
          ymin = ymax = y1;
          ymin = ( y2 < ymin ) ? y2 : ymin;
          ymin = ( y3 < ymin ) ? y3 : ymin;
          ymax = ( y2 > ymax ) ? y2 : ymax;
          ymax = ( y3 > ymax ) ? y3 : ymax;

          /* Determine minimum and maximum x-coordinates */
          xmin = xmax = x1;
          xmin = ( x2 < xmin ) ? x2 : xmin;
          xmin = ( x3 < xmin ) ? x3 : xmin;
          xmax = ( x2 > xmax ) ? x2 : xmax;
          xmax = ( x3 > xmax ) ? x3 : xmax;

          /* Calculate line coordinates for the triangle sides */
          buf12 = bresenham( x1, y1, x2, y2 );
          buf23 = bresenham( x2, y2, x3, y3 );
          buf13 = bresenham( x1, y1, x3, y3 );

          /* Build arrays for x values at all possible y values */
          numy = ymax - ymin + 1;
          xleft = (int *)malloc( (size_t)( numy * 2 ));
          xright = (int *)malloc( (size_t)( numy * 2 ));

          /* Fill arrays with starting values */
          for ( i = 0; i < numy; i++ )
             {
             xleft[i] = xmax;
             xright[i] = xmin;
             }

          /* Put coordinates for first triangle side into arrays */
          for ( i = 0; i < buf12[0]; i++ )
             {
             x = buf12[i+i+1];
             y = buf12[i+i+2] - ymin;
             if ( x < xleft[y] )
                    xleft[y] = x;
             if ( x > xright[y] )
                    xright[y] = x;
             }
```

```
            /* Put coordinates for second triangle side into arrays */
            for ( i = 0; i < buf23[0]; i++ )
               {
               x = buf23[i+i+1];
               y = buf23[i+i+2] - ymin;
               if ( x < xleft[y] )
                        xleft[y] = x;
               if ( x > xright[y] )
                        xright[y] = x;
               }

            /* Put coordinates for third triangle side into arrays */
            for ( i = 0; i < buf13[0]; i++ )
               {
               x = buf13[i+i+1];
               y = buf13[i+i+2] - ymin;
               if ( x < xleft[y] )
                        xleft[y] = x;
               if ( x > xright[y] )
                        xright[y] = x;
               }

            /* Now we can fill the triangle efficiently */
            for ( i = 0; i < numy; i++ )
               {
               _moveto( xleft[i], ymin + i );
               _lineto( xright[i], ymin + i );
               }
            /* Free some memory */
            free( buf12 );
            free( buf23 );
            free( buf13 );
            free( xleft );
            free( xright );
            }
}
```

```
/* ---------------------------------------------------
   Name:         GETKEY.H
   Type:         Include
   Language:     Microsoft QuickC
   Demonstrated: GETKEY.C  GETKTEST.C
   Description:  Prototypes and definitions for GETKEY.C
   ---------------------------------------------------
*/

#ifndef GETKEY_DEFINED

#define KEY_F1        15104
#define KEY_F2        15360
#define KEY_F3        15616
#define KEY_F4        15872
#define KEY_F5        16128
#define KEY_F6        16384
#define KEY_F7        16640
#define KEY_F8        16896
#define KEY_F9        17152
#define KEY_F10       17408
#define KEY_SHIFT_F1  21504
#define KEY_SHIFT_F2  21760
#define KEY_SHIFT_F3  22016
#define KEY_SHIFT_F4  22272
#define KEY_SHIFT_F5  22528
#define KEY_SHIFT_F6  22784
#define KEY_SHIFT_F7  23040
#define KEY_SHIFT_F8  23296
#define KEY_SHIFT_F9  23552
#define KEY_SHIFT_F10 23808
#define KEY_CTRL_F1   24064
#define KEY_CTRL_F2   24320
#define KEY_CTRL_F3   24576
#define KEY_CTRL_F4   24832
#define KEY_CTRL_F5   25088
#define KEY_CTRL_F6   25344
#define KEY_CTRL_F7   25600
#define KEY_CTRL_F8   25856
#define KEY_CTRL_F9   26112
#define KEY_CTRL_F10  26368
#define KEY_ALT_F1    26624
#define KEY_ALT_F2    26880
#define KEY_ALT_F3    27136
#define KEY_ALT_F4    27392
#define KEY_ALT_F5    27648
#define KEY_ALT_F6    27904
#define KEY_ALT_F7    28160
#define KEY_ALT_F8    28416
#define KEY_ALT_F9    28672
#define KEY_ALT_F10   28928
#define KEY_INSERT    20992
#define KEY_HOME      18176
#define KEY_PGUP      18688
#define KEY_DELETE    21248
#define KEY_END       20224
#define KEY_PGDN      20736
#define KEY_UP        18432
#define KEY_LEFT      19200
#define KEY_DOWN      20480
#define KEY_RIGHT     19712
#define KEY_ENTER     13
#define KEY_ESCAPE    27
#define KEY_BACKSPACE 8
#define KEY_TAB       9
#define KEY_SHIFT_TAB 3840
#define KEY_CTRL_LEFT 29440
#define KEY_CTRL_RIGHT 29696
#define KEY_CTRL_HOME 30464
```

```
#define KEY_CTRL_PGUP  33792
#define KEY_CTRL_PGDN  30208
#define KEY_CTRL_END   29952
#define KEY_CTRL_ENTER 10

unsigned int getkey( void );
unsigned int getkey_or_mouse( void );
long student_timer( int *key, char *neutral, unsigned timeout, unsigned warning_time );

#define GETKEY_DEFINED
#endif
```

```
/* ---------------------------------------------------
   Name          MOUSEFUN H
   Type:         Include
   Language      Microsoft QuickC version 2
   Demonstrated  MOUSEFUN C  MOUSTEST C
   Description:  Prototypes and definitions for MOUSEFUN C
   ---------------------------------------------------
*/

#ifndef MOUSEFUN_DEFINED

#define LBUTTON 0
#define RBUTTON 1

#define SOFT_TEXT_CURSOR 0
#define HARD_TEXT_CURSOR 1

#define ENGLISH 0
#define FRENCH 1
#define DUTCH 2
#define GERMAN 3
#define SWEDISH 4
#define FINNISH 5
#define SPANISH 6
#define PORTUGESE 7
#define ITALIAN 8

#define MOUSE_BUS 1
#define MOUSE_SERIAL 2
#define MOUSE_INPORT 3
#define MOUSE_PS2 4
#define MOUSE_HP 5

#define IRQ_PS2 0

/* Structure definition for graphics mode mouse cursors */
struct graphics_cursor
   {
   int screen_mask[16],
   int cursor_mask[16],
   int hot_spot_x,
   int hot_spot_y,
   },

void mouse_reset( int *, int * ),                          /* Function  0 */
void mouse_show( void ),                                   /* Function  1 */
void mouse_hide( void ),                                   /* Function  2 */
void mouse_status( int *, int *, int *, int * ),           /* Function  3 */
void mouse_setpos( int, int ),                             /* Function  4 */
void mouse_press( int, int *, int *, int *, int * ),       /* Function  5 */
void mouse_release( int, int *, int *, int *, int * ),     /* Function  6 */
void mouse_sethorz( int, int ),                            /* Function  7 */
void mouse_setvert( int, int ),                            /* Function  8 */
void mouse_setgcurs( struct graphics_cursor far * ),       /* Function  9 */
void mouse_settcurs( int, int, int ),                      /* Function 10 */
void mouse_motion( int *, int * ),                         /* Function 11 */
void mouse_setratios( int, int ),                          /* Function 15 */
void mouse_condoff( int, int, int, int ),                  /* Function 16 */
void mouse_setdouble( int ),                               /* Function 19 */
void mouse_storage( int * ),                               /* Function 21 */
void mouse_save( char far * ),                             /* Function 22 */
void mouse_restore( char far * ),                          /* Function 23 */
void mouse_setsensitivity( int, int, int ),                /* Function 26 */
void mouse_getsensitivity( int *, int *, int * ),          /* Function 27 */
void mouse_setmaxrate( int ),                              /* Function 28 */
void mouse_setpage( int ),                                 /* Function 29 */
void mouse_getpage( int * ),                               /* Function 30 */
void mouse_setlang( int ),                                 /* Function 34 */
void mouse_getlang( int * ),                               /* Function 35 */
```

```
void mouse_getversion( double *, int *, int * );          /* Function 36 */


/* Default graphics mode cursor */
static struct graphics_cursor far gcursor_default =
   {

   /*  screen mask  */
   0xCFFF,   /* 1100111111111111 */
   0xC7FF,   /* 1100011111111111 */
   0xC3FF,   /* 1100001111111111 */
   0xC1FF,   /* 1100000111111111 */
   0xC0FF,   /* 1100000011111111 */
   0xC07F,   /* 1100000001111111 */
   0xC03F,   /* 1100000000111111 */
   0xC01F,   /* 1100000000011111 */
   0xC00F,   /* 1100000000001111 */
   0xC007,   /* 1100000000000111 */
   0xC07F,   /* 1100000001111111 */
   0xC43F,   /* 1100010000111111 */
   0xCC3F,   /* 1100110000111111 */
   0xFE1F,   /* 1111111000011111 */
   0xFE1F,   /* 1111111000011111 */
   0xFF1F,   /* 1111111100011111 */

   /*  cursor mask  */
   0x0000,   /* 0000000000000000 */
   0x1000,   /* 0001000000000000 */
   0x1800,   /* 0001100000000000 */
   0x1C00,   /* 0001110000000000 */
   0x1E00,   /* 0001111000000000 */
   0x1F00,   /* 0001111100000000 */
   0x1F80,   /* 0001111110000000 */
   0x1FC0,   /* 0001111111000000 */
   0x1FE0,   /* 0001111111100000 */
   0x1F00,   /* 0001111100000000 */
   0x1B00,   /* 0001101100000000 */
   0x1180,   /* 0001000110000000 */
   0x0180,   /* 0000000110000000 */
   0x00C0,   /* 0000000011000000 */
   0x00C0,   /* 0000000011000000 */
   0x0000,   /* 0000000000000000 */

   /* hot spot x,y */
   02, 00
   };


/* Graphics mode cursor, pointing hand */
static struct graphics_cursor far gcursor_hand =
   {

   /*  screen mask  */
   0xE1FF,   /* 11100h0111111111 */
   0xE1FF,   /* 1110000111111111 */
   0xE1FF,   /* 1110000111111111 */
   0xE1FF,   /* 1110000111111111 */
   0xE1FF,   /* 1110000111111111 */
   0xE000,   /* 1110000000000000 */
   0xE000,   /* 1110000000000000 */
   0xE000,   /* 1110000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
```

```
    /* cursor mask */
    0x1E00,   /* 00011H1000000000 */
    0x1200,   /* 0001001000000000 */
    0x1200,   /* 0001001000000000 */
    0x1200,   /* 0001001000000000 */
    0x1200,   /* 0001001000000000 */
    0x13FF,   /* 0001001111111111 */
    0x1249,   /* 0001001001001001 */
    0x1249,   /* 0001001001001001 */
    0xF249,   /* 1111001001001001 */
    0x9001,   /* 1001000000000001 */
    0x9001,   /* 1001000000000001 */
    0x9001,   /* 1001000000000001 */
    0x8001,   /* 1000000000000001 */
    0x8001,   /* 1000000000000001 */
    0x8001,   /* 1000000000000001 */
    0xFFFF,   /* 1111111111111111 */

    /* hot spot x,y */
    05, 00
    };


/* Graphics mode cursor, check mark */
static struct graphics_cursor far gcursor_check =
    {

    /* screen mask */
    0xFFF0,   /* 1111111111110000 */
    0xFFE0,   /* 1111111111100000 */
    0xFFC0,   /* 1111111111000000 */
    0xFF81,   /* 1111111110000001 */
    0xFF03,   /* 1111111100000011 */
    0x0607,   /* 0000011000000111 */
    0x000F,   /* 0000000000001111 */
    0x001F,   /* 0000000000011111 */
    0xC03F,   /* 1100000000111111 */
    0xF07F,   /* 1111000001111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */

    /* cursor mask */
    0x0000,   /* 0000000000000000 */
    0x0006,   /* 0000000000000110 */
    0x000C,   /* 0000000000001100 */
    0x0018,   /* 0000000000011000 */
    0x0030,   /* 0000000000110000 */
    0x0060,   /* 0000000001100000 */
    0x70C0,   /* 0111000011000000 */
    0x1D80,   /* 0001110110000000 */
    0x0700,   /* 0000011100000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */

    /* hot spot x,y */
    06, 07
    };
```

```
/* Graphics mode cursor, hour glass */
static struct graphics_cursor far gcursor_hour =
   {

   /* screen mask */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x8001,   /* 1000000000000001 */
   0xC003,   /* 1100000000000011 */
   0xE007,   /* 1110000000000111 */
   0xF00F,   /* 1111000000001111 */
   0xE007,   /* 1110000000000111 */
   0xC003,   /* 1100000000000011 */
   0x8001,   /* 1000000000000001 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */

   /* cursor mask */
   0x0000,   /* 0000000000000000 */
   0x7FFE,   /* 0111111111111110 */
   0x6006,   /* 0110000000000110 */
   0x300C,   /* 0011000000001100 */
   0x1818,   /* 0001100000011000 */
   0x0C30,   /* 0000110000110000 */
   0x0660,   /* 0000011001100000 */
   0x03C0,   /* 0000001111000000 */
   0x0660,   /* 0000011001100000 */
   0x0C30,   /* 0000110000110000 */
   0x1998,   /* 0001100110011000 */
   0x33CC,   /* 0011001111001100 */
   0x67E6,   /* 0110011111100110 */
   0x7FFE,   /* 0111111111111110 */
   0x0000,   /* 0000000000000000 */
   0x0000,   /* 0000000000000000 */

   /* hot spot x,y */
   07, 07
   };


/* Graphics mode cursor, jet aircraft */
static struct graphics_cursor far gcursor_jet =
   {

   /* screen mask */
   0xFFFF,   /* 1111111111111111 */
   0xFEFF,   /* 1111111011111111 */
   0xFC7F,   /* 1111110001111111 */
   0xF83F,   /* 1111100000111111 */
   0xF83F,   /* 1111100000111111 */
   0xF83F,   /* 1111100000111111 */
   0xF01F,   /* 1111000000011111 */
   0xE00F,   /* 1110000000001111 */
   0xC007,   /* 1100000000000111 */
   0x8003,   /* 1000000000000011 */
   0x8003,   /* 1000000000000011 */
   0xF83F,   /* 1111100000111111 */
   0xF83F,   /* 1111100000111111 */
   0xF01F,   /* 1111000000011111 */
   0xE00F,   /* 1110000000001111 */
   0xFFFF,   /* 1111111111111111 */

   /* cursor mask */
   0x0000,   /* 0000000000000000 */
```

```
    0x0000,   /* 0000000000000000 */
    0x0100,   /* 0000000100000000 */
    0x0380,   /* 0000001110000000 */
    0x0380,   /* 0000001110000000 */
    0x0380,   /* 0000001110000000 */
    0x07C0,   /* 0000011111000000 */
    0x0FE0,   /* 0000111111100000 */
    0x1FF0,   /* 0001111111110000 */
    0x3FF8,   /* 0011111111111000 */
    0x638C,   /* 0110001110001100 */
    0x0380,   /* 0000001110000000 */
    0x0380,   /* 0000001110000000 */
    0x07C0,   /* 0000011111000000 */
    0x0C60,   /* 0000110001100000 */
    0x0000,   /* 0000000000000000 */

    /* hot spot x,y */
    07, 01
    };


/* Graphics mode cursor, left pointing arrow */
static struct graphics_cursor far gcursor_left =
    {

    /*  screen mask  */
    0xFE1F,   /* 1111111000011111 */
    0xF01F,   /* 1111000000011111 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0xF01F,   /* 1111000000011111 */
    0xFE1F,   /* 1111111000011111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */

    /*  cursor mask  */
    0x0000,   /* 0000000000000000 */
    0x00C0,   /* 0000000011000000 */
    0x07C0,   /* 0000011111000000 */
    0x7FFE,   /* 0111111111111110 */
    0x07C0,   /* 0000011111000000 */
    0x00C0,   /* 0000000011000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */

    /* hot spot x,y */
    00, 03
    };


/* Graphics mode cursor, plus sign */
static struct graphics_cursor far gcursor_plus =
    {
```

```
  /* screen mask */
  0xFC3F,   /* 1111110000111111 */
  0xFC3F,   /* 1111110000111111 */
  0xFC3F,   /* 1111110000111111 */
  0x0000,   /* 0000000000000000 */
  0x0000,   /* 0000000000000000 */
  0x0000,   /* 0000000000000000 */
  0xFC3F,   /* 1111110000111111 */
  0xFC3F,   /* 1111110000111111 */
  0xFC3F,   /* 1111110000111111 */
  0xFFFF,   /* 1111111111111111 */
  0xFFFF,   /* 1111111111111111 */
  0xFFFF,   /* 1111111111111111 */
  0xFFFF,   /* 1111111111111111 */
  0xFFFF,   /* 1111111111111111 */
  0xFFFF,   /* 1111111111111111 */
  0xFFFF,   /* 1111111111111111 */

  /* cursor mask */
  0x0000,   /* 0000000000000000 */
  0x0180,   /* 0000000110000000 */
  0x0180,   /* 0000000110000000 */
  0x0180,   /* 0000000110000000 */
  0x7FFE,   /* 0111111111111110 */
  0x0180,   /* 0000000110000000 */
  0x0180,   /* 0000000110000000 */
  0x0180,   /* 0000000110000000 */
  0x0000,   /* 0000000000000000 */
  0x0000,   /* 0000000000000000 */
  0x0000,   /* 0000000000000000 */
  0x0000,   /* 0000000000000000 */
  0x0000,   /* 0000000000000000 */
  0x0000,   /* 0000000000000000 */
  0x0000,   /* 0000000000000000 */

  /* hot spot x,y */
  07, 04
  };

/* Graphics mode cursor, up pointing arrow */
static struct graphics_cursor far gcursor_up =
  {

  /* screen mask */
  0xF9FF,   /* 1111100111111111 */
  0xF0FF,   /* 1111000011111111 */
  0xE07F,   /* 1110000001111111 */
  0xE07F,   /* 1110000001111111 */
  0xC03F,   /* 1100000000111111 */
  0xC03F,   /* 1100000000111111 */
  0x801F,   /* 1000000000011111 */
  0x801F,   /* 1000000000011111 */
  0x000F,   /* 0000000000001111 */
  0x000F,   /* 0000000000001111 */
  0xF0FF,   /* 1111000011111111 */
  0xF0FF,   /* 1111000011111111 */
  0xF0FF,   /* 1111000011111111 */
  0xF0FF,   /* 1111000011111111 */
  0xF0FF,   /* 1111000011111111 */
  0xF0FF,   /* 1111000011111111 */

  /* cursor mask */
  0x0000,   /* 0000000000000000 */
  0x0600,   /* 0000011000000000 */
  0x0F00,   /* 0000111100000000 */
  0x0F00,   /* 0000111100000000 */
```

```
    0x1F80,   /* 0001111110000000 */
    0x1F80,   /* 0001111110000000 */
    0x3FC0,   /* 0011111111000000 */
    0x3FC0,   /* 0011111111000000 */
    0x7FE0,   /* 0111111111100000 */
    0x0600,   /* 0000011000000000 */
    0x0600,   /* 0000011000000000 */
    0x0600,   /* 0000011000000000 */
    0x0600,   /* 0000011000000000 */
    0x0600,   /* 0000011000000000 */
    0x0600,   /* 0000011000000000 */
    0x0000,   /* 0000000000000000 */

    /* hot spot x,y */
    05, 00
    };


/* Graphics mode cursor, X mark */
static struct graphics_cursor far gcursor_x =
    {

    /*  screen mask  */
    0x07E0,   /* 0000011111100000 */
    0x0180,   /* 0000000110000000 */
    0x0000,   /* 0000000000000000 */
    0xC003,   /* 1100000000000011 */
    0xF00F,   /* 1111000000001111 */
    0xC003,   /* 1100000000000011 */
    0x0000,   /* 0000000000000000 */
    0x0180,   /* 0000000110000000 */
    0x03C0,   /* 0000001111000000 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */
    0xFFFF,   /* 1111111111111111 */

    /*  cursor mask  */
    0x0000,   /* 0000000000000000 */
    0x700E,   /* 0111000000001110 */
    0x1C38,   /* 0001110000111000 */
    0x0660,   /* 0000011001100000 */
    0x03C0,   /* 0000001111000000 */
    0x0660,   /* 0000011001100000 */
    0x1C38,   /* 0001110000111000 */
    0x700E,   /* 0111000000001110 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */
    0x0000,   /* 0000000000000000 */

    /* hot spot x,y */
    07, 04
    };

#define MOUSEFUN_DEFINED
#endif
```

```
/* ---------------------------------------------------------

        Name:       VIDEO.H
        Type:       Include
        Language:   Microsoft QuickC version 2
        Description:  Prototypes and definitions for VIDEO.C

        ---------------------------------------------------------
*/

#ifndef VIDEO_DEFINED

#define FALSE 0
#define TRUE !FALSE

#define LINED 0
#define SOLID 1


int *bresenham( int, int, int, int );
void triangle(int, int, int, int, int, int, int);
void polygon( int, int, int [][2] );
void line( int x1, int y1, int x2, int y2 );

/* define structures */
struct points
{
        short x;
        short y;
};

/* Define functions */
void text_mode( void );
void best_graph_mode( void );
double x_factor( void );
double y_factor( void );
int maximum_y( void );
int device_x( int );
int device_y( int );
void line( int, int, int, int );
void saveimage( int, int, int, int );
void restimage( void );
double k_time( int *key );
double m_time( int *key );
double time_3( int *key );

#define VIDEO_DEFINED
#endif
```

```
/* ---------------------------------------------------

        Name:       TYPE_INIT.H
        Type:       Include
        Language:   Microsoft QuickC version 2
        Description:  Prototypes and definitions for use
                                        with various modules used by SECURE.C
 ---------------------------------------------------
*/

#pragma pack(1)

/* Type defintions */

typedef struct {
        char qualifier[10];
        long offset;
} INDEX_INFO;


/*
 *   Student Column #      Test#      Trial#
 *
 *       0            1        1
 *       1            2        1
 *       2            2        2
 *       3            2        3
 *       4            2        4
 *       5            2        5
 *       6            2        6
 *       7            3        1
 *
 */
typedef struct {
        double  avg_time_correct;
        double  ovrl_avg_time_corr;
        double  avg_time_incorrect;
        double  ovrl_avg_time_incorr;
        int   no_questions_correct;
        int   total_no_questions;
} STUDENT_COLUMN;

typedef struct {
        double reaction_time;          /* reaction time to question      */
        char   answer;                 /* answer to question             */
        char   right_wrong;            /* 1 if correct answer, 0 if incorrect */
} TEMP;

typedef struct {
        int   test_no;
        char  qualifier[10];
        char  r_l_handed;
        char  male_female;
        STUDENT_COLUMN student_info[10];
        TEMP RESPONSE[65]; /* 64 problems in test 1 */
} STUDENT_RECORD;

typedef struct qualifier_rec {
        char    qualifier[10];
        long    offset;
        struct  qualifier_rec *next;
} NODE;
typedef struct results_rec {
        int     test_no;
        char    qualifier[10];
        char    r_l_handed;
        STUDENT_COLUMN student_info[30];
        struct  results_rec *next;
} RES_NODE;
```

```
/* ---------------------------------------------------
        Name:       TMANAGER.H
  Type:       Include
  Language:     Microsoft QuickC version 2
        Description:  Definition of functions for test manager
  ---------------------------------------------------
*/

/* Function definitions */
void Test_manager( STUDENT_RECORD *new_student );
```

```
/* ---------------------------------------------------

        Name:       MN_MENU.H
        Type:       Include
        Language:     Microsoft QuickC version 2
        Description:  Prototypes and definitions for MN_MENU.C

        ---------------------------------------------------
*/

/* Define functions */
void display_main_menu( NODE **, NODE **, RES_NODE **, RES_NODE ** );
```

```
/* ---------------------------------------------------

        Name        DSK_INIT H
        Type        Include
        Language    Microsoft QuickC version 2
        Description Prototypes and definitions for DSK_INIT H

        ---------------------------------------------------
*/

/* Define functions */
void Initialize( NODE **, NODE ** ),
void Stats_initialize( RES_NODE **, RES_NODE ** ),

 /* ---------------------------------------------------

        Name        DATA_PLT H
        Type        Include
        Language    Microsoft QuickC version 2
        Description Prototypes and definitions for DATA_PLT C

        ---------------------------------------------------
 */


 char right_or_left_handed( void ),
 char Male_or_female( void ),
 void get_student_data( NODE *, STUDENT_RECORD *, long * ),

/* ---------------------------------------------------
  Name        BOX H
  Type        Include
  Language    Microsoft QuickC version 2
  Description Prototypes and definitions for BOX C
  ---------------------------------------------------
*/

#ifndef BOX_DEFINED

unsigned far *box_get( unsigned, unsigned, unsigned, unsigned ),
void box_put( unsigned far * ),
void box_color( unsigned, unsigned, unsigned, unsigned ),
void box_charfill( unsigned, unsigned, unsigned, unsigned, unsigned char ),
void box_draw( unsigned, unsigned, unsigned, unsigned, unsigned ),
void box_erase( unsigned, unsigned, unsigned, unsigned ),

#define BOX_DEFINED
#endif

 /* ---------------------------------------------------

        Name        MK_FP H
        Type        Include
        Language    Microsoft QuickC version 2
        Description Macro to form a far pointer
        ---------------------------------------------------
 */

 #define MK_FP( seg, off ) (( void far * ) \
                            ((( unsigned long )( seg ) << 16 ) + ( unsigned )( off )))
```

```
/* ---------------------------------------------------

        Name:        STATS.H
        Type:        Include
        Language:    Microsoft QuickC version 2
        Description: Prototypes and definitions for STATS.C

        ---------------------------------------------------
*/

/* Define functions */
double cal_mean_time_correct( int, RES_NODE * );
double cal_mean_time_incorrect( int, RES_NODE * );
double cal_stat_deviation_correct( int, RES_NODE * );
double cal_stat_deviation_incorrect( int, RES_NODE * );
void stats_test_1( TEMP *, STUDENT_RECORD *, int *, int );
void stats_test_2( TEMP *, STUDENT_RECORD *, int *, int );
void stats_test_3( TEMP *, STUDENT_RECORD *, char *, int, int );
void stats_test_4( TEMP *, STUDENT_RECORD *, int[], int, int );
void Get_mtc_data( float *, RES_NODE * );
void Get_mti_data( float *, RES_NODE * );
void Get_pc_data( float *, RES_NODE * );
void mean_time_correct( float *, RES_NODE * );

/* ---------------------------------------------------
   Name:        SOUND.H
   Type:        Include
   Language:    Microsoft QuickC
   Description: Prototypes and definitions for SOUND.C
   ---------------------------------------------------
*/

#ifndef SOUND_DEFINED

void sound( int );
void silence( void );
void speaker_toggle( void );
void wait_ticks( unsigned int );
void warble( int );
void weird( int );
void siren( int );
void white_noise( int );
void note( int, int );

#define SOUND_DEFINED
#endif

/* ---------------------------------------------------

        Name:        FILE.H
        Type:        Include
        Language:    Microsoft QuickC version 2
        Description: Prototypes and definitions for FILE.C

        ---------------------------------------------------
*/

/* Defines */
#define FILENAME  "STUDENT.FIL"
#define INDEX     "STUDENT.NDX"
#define TEMP      "STUDENT.TMP"

/* Define functions */
int  Index_on_disk( void );
int  File_on_disk( void );
int  Num_records( void );
int  Index_to_link_list( int, NODE **, NODE ** );
void Fetch( long, STUDENT_RECORD * );
```

```
/* ---------------------------------------------------
   Name         EDIT H
   Type         Include
   Language     Microsoft QuickC version 2
   Demonstrated   EDIT C  EDITTEST C
   Description    Prototypes and definitions for EDIT
   ---------------------------------------------------
*/

#ifndef EDIT_DEFINED

#define CURSOR_UNDERLINE  0x0707
#define CURSOR_BLOCK      0x0007
#define CURSOR_DOUBLELINE 0x0607
#define CURSOR_NONE       0x2000

int next_word( char *, int ),
int prev_word( char *, int ),
int delete_char( char *, int ),
int insert_char( char *, int, char ),
int insert_spaces( char *, int, int ),
int replace( char *, char *, char * ),
int editline( char * ),

#define EDIT_DEFINED
#endif
/* ---------------------------------------------------
   Name         MENU H
   Type         Include
   Language     Microsoft QuickC
   Demonstrated   MENU C  MENUTEST C
   Description    Prototypes and definitions for MENU module
   ---------------------------------------------------
*/

#ifndef MENU_DEFINED

void menu_box_lines( int ),
void menu_box_shadow( int ),
void menu_back_color( long int ),
void menu_line_color( int ),
void menu_title_color( int ),
void menu_text_color( int ),
void menu_prompt_color( int ),
void menu_hilight_letter( int ),
void menu_hilight_text( int ),
void menu_hilight_back( long int ),
int far *menu_bar( int, int, char *, int * ),
int far *menu_drop( int, int, char **, int * ),
int far *menu_message( int, int, char ** ),
void menu_erase( int far * ),

#define MENU_DEFINED
#endif
```

```
/* ---------------------------------------------------

        Name:       T1OBJECTS.H
        Type:       Include
        Language:   Microsoft QuickC version 2
        Description: Prototypes and definitions for T1OBJECTS.C

        ---------------------------------------------------
*/

#ifndef T1OBJECTS_DEFINED

/* Define functions */
void Draw_background( void );
void Draw_example_background( void );
void Draw_plane( float heading );
void Draw_aircraft_problem( short ac_orientation, short ac_position );
void Draw_example_aircraft_problem( short ac_orientation, short ac_position );
void Init_ac_orientations( void );
void Free_ac( void );
void press_key( void );
void example_sound_prompt( void );
void text_bar( void );
void mid_text_bar( void );
void down_text_bar( void );
void up_black_bar( void );
void custom_bar( int x1, int y1, int x2, int y2, int color );
void print_countdown( void );
void begin_message( void );
void Dash_line( int xcoord_1, int ycoord_1,
            int xcoord_2, int ycoord_2, int num_dashes);
void display_test_name( char *test_name );

#define T1OBJECTS_DEFINED
#endif
 /* ---------------------------------------------------
   Name:       T_COLORS.H
   Type:       Include
   Language:   Microsoft QuickC version 2
   Demonstrated: BOXTEST.C  COLORS.C  EDITTEST.C
                       MENU.C  LOOK.C  OBJECT.C
   Description: Definitions for text mode color constants
   ---------------------------------------------------
 */

 #ifndef T_COLORS_DEFINED

 /* Standard text mode colors */
 #define T_BLACK 0
 #define T_BLUE 1
 #define T_GREEN 2
 #define T_CYAN 3
 #define T_RED 4
 #define T_MAGENTA 5
 #define T_BROWN 6
 #define T_WHITE 7

 /* Modifiers that can be added to the text mode color constants */
 #define T_BRIGHT 8
 #define T_BLINK 16

 /* Common combinations */
 #define T_GRAY ( T_BLACK | T_BRIGHT )
 #define T_YELLOW ( T_BROWN | T_BRIGHT )

 /* Background text mode color constants */
 #define BK_BLACK 0L
 #define BK_BLUE 1L
 #define BK_GREEN 2L
```

```
/* ---------------------------------------------------------

        Name        LIST H
        Type        Include
        Language    Microsoft QuickC version 2
        Description Prototypes and definitions for LIST C

        ---------------------------------------------------------
*/


/*
        Routines are used to load information held in index
        file into linked list  Linked list is for determining
        which students the system has test data

        Define functions for manipulating nodes of type NODE
*/

void addsl( long, NODE **, NODE **, char * ),
void freelist( NODE * ),
long check( NODE *, char * ),


/*
        Routines are used to load information held in student
        data file into linked list  Linked list is for used
        for statistical manipulation of test data

        Define functions for manipulating nodes of type RES_NODE
*/

void res_addsl( STUDENT_RECORD *, RES_NODE **, RES_NODE ** ),
void res_freelist( RES_NODE * ),
```

```
PROJ    =ROTATE
DEBUG   =1
CC      =qcl
AS      =qcl
CFLAGS_G        = /AL /W1 /Ze
CFLAGS_D        = /Zd /Zr /Gi$(PROJ).mdt /Od
CFLAGS_R        = /O /Ot /Gs /DNDEBUG
CFLAGS  =$(CFLAGS_G) $(CFLAGS_D)
AFLAGS_G        = /Cx /W1 /P2
AFLAGS_D        = /Zd
AFLAGS_R        = /DNDEBUG
AFLAGS  =$(AFLAGS_G) $(AFLAGS_D)
LFLAGS_G        = /CP:0xffff /NOI /SE:0x80 /ST:0x1000
LFLAGS_D        =
LFLAGS_R        =
LFLAGS  =$(LFLAGS_G) $(LFLAGS_D)
RUNFLAGS        =
OBJS_EXT =
LIBS_EXT =

.asm.obj:
        $(AS) $(AFLAGS) -c $*.asm

all:    $(PROJ).EXE

rotate.obj:     rotate.c $(H)

data_plt.obj:   data_plt.c $(H)

dsk_init.obj:   dsk_init.c $(H)

edit.obj:       edit.c $(H)

file.obj:       file.c $(H)

getkey.obj:     getkey.c $(H)

list.obj:       list.c $(H)

menu.obj:       menu.c $(H)

mn_menu.obj:    mn_menu.c $(H)

mousefun.obj:   mousefun.c $(H)

sound.obj:      sound.c $(H)

stats.obj:      stats.c $(H)

t1object.obj:   t1object.c $(H)

test_1.obj:     test_1.c $(H)

tmanager.obj:   tmanager.c $(H)

video.obj:      video.c $(H)

box.obj:        box.c $(H)

$(PROJ).EXE:    rotate.obj data_plt.obj dsk_init.obj edit.obj file.obj getkey.obj list.obj \
                menu.obj mn_menu.obj mousefun.obj sound.obj stats.obj t1object.obj test_1.obj tmanager.obj \
                video.obj box.obj $(OBJS_EXT)
                echo >NUL @<<$(PROJ).crf
rotate.obj +
data_plt.obj +
dsk_init.obj +
edit.obj +
file.obj +
getkey.obj +
list.obj +
```

```
menu.obj +
mn_menu.obj +
mousefun.obj +
sound.obj +
stats.obj +
t1object.obj +
test_1.obj +
tmanager.obj +
video.obj +
box.obj +
$(OBJS_EXT)
$(PROJ).EXE

$(LIBS_EXT);
<<
        ilink -a -e "qlink $(LFLAGS) @$(PROJ).crf" $(PROJ)

run: $(PROJ).EXE
        $(PROJ) $(RUNFLAGS)
```