THE DESIGN OF A PC SOFTWARE PACKAGE
FOR A PREVIEW TRACKING TASK
USING BORLAND C++ AND WINDOWS/

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

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

by
Jui-Lin Chen
June 1994
ACKNOWLEDGMENT

I wish to express my deep appreciation to my thesis advisor Dr. Helmut T. Zwahlen for his guidance, assistance and encouragement leading to the successful completion of the thesis project.

Special thanks to Dr. Robert Williams and Dr. Harry M. Kaneshige for their serving on the committee and giving valuable suggestions. I also like to thank my friend, Mr. Thomas Schnell, for his help of developing the installation program of this software package which is the core of my thesis.
# TABLE OF CONTENTS

1. INTRODUCTION .......................................................... 1
   1.1 Background .......................................................... 1
   1.2 Statement of problem .............................................. 4
   1.3 Literature review .................................................. 6
       1.3.1 Human tracking performance ................................. 6
       1.3.2 Digital pursuit and compensatory control ............... 7
       1.3.3 Driver steering control and performance ................. 7
       1.3.4 Preview tracking experiments .............................. 8
       1.3.5 Software package for a preview tracking task .......... 10
   1.4 Considerable Experimental Factors ............................. 12

2. CONCEPTUAL DESIGN .................................................... 15
   2.1 Objective ........................................................... 15
   2.2 Basic concepts .................................................... 15
   2.3 Requirements ....................................................... 17
       2.3.1 Hardware requirements ....................................... 17
       2.3.2 Software requirements ....................................... 19
       2.3.3 Ergonomic requirements ..................................... 21
       2.3.4 Experimenter requirements .................................. 21
       2.3.5 Subject requirements ......................................... 22
       2.3.6 System requirements ......................................... 22
   2.4 Constraints .......................................................... 23

3. PRELIMINARY DESIGN .................................................... 25
   3.1 System operation functions of SIMULATOR ..................... 28
   3.2 System operation functions of TRACER ......................... 43
   3.3 Resource definition files ......................................... 49
   3.4 Module definition files ........................................... 50

4. DETAILED DESIGN ....................................................... 52
   4.1 Screen display of SIMULATOR ..................................... 52
   4.2 Algorithm of road generation .................................... 55
   4.3 Input device control routine ..................................... 63
   4.4 Erase and redraw routine ......................................... 70
   4.5 Off road checking .................................................. 75
   4.6 Storing experimental data ........................................ 75
   4.7 Data protection in Multiple Document Interface (MDI) ........ 77
   4.8 Printing ............................................................. 78
5. TEST AND VALIDATION ........................................... 80
  5.1 A simple experiment conducted to a practicability test ......................... 80
  5.1.1 Experimental design and procedure ........................................... 80
  5.1.2 Experimental results ..................................................... 82
  5.2 Validation of relative curve frequency ......................................... 87
  5.3 Validation of mean value and standard deviation ................................ 89

6. CONCLUSIONS .......................................................... 91
  6.1 Suggestions for the future research ............................................. 92

BIBLIOGRAPHY ............................................................. 94

APPENDICES:
  A. SOURCE CODE OF SIMULATOR WITH DETAILED DESCRIPTION ....................... 97
  B. SOURCE CODE OF TRACER WITH DETAILED DESCRIPTION .......................... 167
  C. RESOURCE DEFINITION FILE OF SIMULATOR WITH DETAILED DESCRIPTION .... 254
  D. RESOURCE DEFINITION FILE OF TRACER WITH DETAILED DESCRIPTION ........ 264
  E. USER MANUAL ............................................................ 269
  F. DATA STRUCTURE OF .DAT, .INF, AND .RST FILES ........................... 334
LIST OF TABLES

TABLE

4.1 Suggestions and ideas for improving the efficiency of the road generating algorithm.......................... 57
4.2 Algorithm of road generation.......................... 62
5.1 The list of experimental runs.......................... 81
5.2 Experimental results of 6 subjects and 8 experimental conditions............................................. 82
5.2 ANOVA table of the experimental result analysis...... 85
5.4 The comparison of the mean and standard deviation calculated by hands and by analysis package............ 90
LIST OF FIGURES

FIGURE

1.1 Schematic representation of human-machine system ..... 3
1.2 The relationship between preview tracking performance
and some important experimental factors .......... 14
3.1 Function architecture of the whole package ........ 26
3.2 Flowchart of MainWndProc......................... 30 - 32
3.3 Related curve shapes of .INI files ............... 34
3.4 Flowchart of SimmWndProc......................... 36 - 39
3.5 The initial window used in SIMULATOR and TRACER .... 42
3.6 Flowchart of FrameWndProc........................ 45 - 47
4.1 The coordinate system used in developing the package. 54
4.2 The screen design of SIMULATOR.................... 56
4.3 The positions of nLX, nCX and nRX ............... 61
4.4 Flowchart of SelectShape......................... 64 - 66
4.5 Flowchart of YokeInput............................ 71 - 72
5.1 Experimental results illustrated in mean value and
standard deviations .................................. 83
5.2 The main effect of three variables ................ 84
5.3 Validation of relative curve rate................... 88
CHAPTER I
INTRODUCTION

Computer simulation was a technique for which understanding of the behavior of a physical system was obtained by making measurements or observations of the behavior of a model representing the system. It was one of the best tools to detect the physical interaction and analytical estimation of various assembled variables of a real world system and the system's external environment. The results from simulations were projections of the expected behavior of the system being modeled. The accuracy of the results was completely dependent upon the nature of the simulation. With the advancement of interactive computer graphics techniques, perceiving the systematic estimates of the simulated situations by visualizing the detailed simulation procedure in combination with graphics and animation becomes an easy-to-reach goal.

1.1 BACKGROUND

The human-machine interactive system was the kernel of research in the human factors field. A typical human-machine system was a combination of several human beings and various physical objects that interacted to achieve the
desired outputs from the given inputs. Those physical objects include tools, equipment, and anything that could carry out some activities that were directed toward reaching the desired purpose. Figure 1.1 illustrated an example of a typical interaction between a person and a machine. It showed how the displays of a machine serve as stimuli for an operator, then triggered some types of information processing on the part of the operator, which in turn resulted in some actions that controlled the operation of the machine.

Tracking was a special form of human-machine interaction, which was involved when the continuous control had to accommodate some external signal inputs. A typical tracking task required information processing and decision-making activities to perform the accurate control of a system at the proper time. There were two types of tracking tasks that were usually used in human factors experiments: pursuit and compensatory tracking. The pursuit tracking display showed the indicated movement of both the controlled element (the output) and the target (the input). The compensatory tracking display showed only the single input, the absolute error, or the difference between the target and the controlled element.
Figure 1.1 Schematic representation of human-machine system

In the summary of Poulton's (1974) studies, the pursuit display was better than the compensatory display when only one choice could be made, because the pursuit display involved the greater movement compatibility. With the better compatibility, users could find the disconnected movements of both the input and the output.

Many tracking tasks offered the subject a preview of the track ahead. This resulted in a true display showing the track from the present time to the future time, and this allowed the subject to compensate for the influence of the time lag. Sheridan and Ferrell (1974) pointed out, that preview tracking control such as driving vehicles on the highway was more representative of an everyday tracking operation than the pursuit control and the compensatory control that involved only two points or one point on a display.

1.2 STATEMENT OF PROBLEM

There were a number of preview tracking experiments that were conducted through the years, such as, Mclean and Hoffman (1973), Jansson and Schenkman (1977), Barth (1979), and D.D. Clark-Carter, A.D. Heyes and C.I. Howarth (1986), etc. The results obtained by their researches will be discussed in the following sections.
However, according to their studies, most of the experimental equipment used in the research was capital-intensive, economically unattainable under limited school funding such as, a real car, the Swedish Laser Cane (Jansson and Schenkman, 1977), an adjustable opaque visor (Barth, 1979), the Sonic Pathfinder (Heyes, 1984), and so on. In addition to this expensive equipment, a specific experimental environment was needed to conduct those experiments. It was very difficult for a school to provide funding for such equipment.

On the other hand, the concept of a preview tracking task was typical to everyday life. It could be seen anywhere, and done at anytime. Therefore, it was highly desirable to create a tool to demonstrate and teach the students in the classroom about the various aspects of the preview tracking task.

Along with the recent improvements of VLSI (Very Large Scale Integration Circuits) and related technology, micro computers had become a very inexpensive tool with better performance to conduct experiments and to teach students. By using today's enhanced graphics computers, it was feasible to develop a software package that simulated a driving/tracking task on the computer screen.
To conclude, the goal of this thesis was to eliminate
the need for very expensive special equipment thus reducing
the need for extended funding by applying the improved
computer technology in order to provide an effective and
inexpensive teaching tool for students in the classroom.

1.3 LITERATURE REVIEW

1.3.1 Human tracking performance

Darvin P. Hunt (1961) conducted a study to determine the
human operator's performance in a compensatory tracking task
as a function of the precision of the displayed error
information for different levels of task difficulty and
practice. The results indicated that as the operator became
more proficient, the relative superiority of the displayed
error with increased precision was reduced.

The effect of the precision of information on control
motion depended on the difficulty of the task. The amount
of control motion generally decreased as the operator became
more proficient, and the magnitude of this decrease was
greater for an easy task than for a difficult task. The
above finding concluded that there was a relationship
between human performance and the precision of feedback
information.
1.3.2 Digital pursuit and compensatory control

There were some researchers who had carried out some experiments comparing human tracking performance when using pursuit and compensatory displays (see Poulton, 1974). The general conclusion based on these studies was that a pursuit display was at least as good as a compensatory display for preview tracking, while for some non-visual displays, a compensatory display seemed to be preferable.

One experiment conducted by Tarald O. Kvalseth (1978) compared human performance tracking for pursuit and compensatory digital displays utilized in a digital control task with a first-order controlled system and a reference input that was either a purely random Gaussian noise or a first-order auto-regressive process. The performance measurements obtained in this study showed no significant differences between the pursuit and compensatory case.

1.3.3 Driver steering control and performance

Driver steering control and performance were studied for straight-lane driving under conditions of restricted far-sight distance (McLean and Hoffman 1973). The far-sight distance necessary for the driver to adequately align the car was found to be 70 ft, and was independent of the vehicle speeds of 20 and 30 m.p.h. A spectral analysis of
the steering wheel angle data showed peaks in the frequency range 0.1 to 0.3 Hz. The value of the peak frequency was affected by the allowed preview time, where the preview time was the far-sight distance divided by vehicle speed. A different approach was taken by McLean and Hoffman (1971) who carried out a spectral analysis of the steering wheel angle obtained during simple driving tasks. They also observed peaks in spectral density in the range of 0.1 to 0.3 Hz, with the value of the peak frequency depending on the driver and the driving task. These peaks indicated that drivers made most of their control movements within a fairly small frequency range.

1.3.4 Preview tracking experiments

The effects of preview as a mean of providing the human operator with anticipate information about the reference input have been analyzed in a number of different manual control studies (see Poluton, 1974). These studies clearly showed the benefits of providing the human operator with input preview during a digital pursuit control task.

One study done by Tarald O. Kvalseth (1978) showed the effect of preview during accurate motor movements aimed at a target. This case was generally referred to as a preview system as opposed to precognitive system for which the
operator did not have a direct and true view of the future input but possessed some statistical foreknowledge of the input.

Another study presented by Kvalseth (1979) examined to what extent the preview input information depended on whether the preview was extended immediately ahead of the present position or was postponed by a certain amount. According to the experiment conducted, a clear design implication for digital man-machine control systems following from this study called for immediate preview of the reference input without any lag, especially when the input was band-limited white noise.

Jansson and Schenkman (1977) used a mobility aid known as the Swedish Laser Cane to present the preview. They found that the longer the range of the cane, the more obstacles were found by the subjects which help them to stop farther away from those obstacles.

One research study conducted by Barth (1979) used subjects whose preview was limited by an adjustable opaque visor. The subjects' horizon of view was restricted by the visor and they were physically prevented from making compensatory head movements. Barth found that the greatest amount of preview allowed subjects to maintain all aspects
of effective mobility except for the mistakes they made at turning points.

1.3.5 Software package for a preview tracking task

A related thesis topic has been researched by C. Viswanathan and Helmut T. Zwahlen (1986). Because of the restricted performance of the Apple IIe computer used, the researchers implemented the preview tracking package in the 6502 assembly language. The package developed by them consisted of six associated modules and an extra experimental variable setup menu coded in the BASIC computer language.

The first module of the package generated specific pixels that needed to be lighted up on the top most line of the screen. This module generated the left curves, the right curves, the tangent sections, and the broken center line. The second module of the tracking package performed the scrolling job. The speed of the car depended upon how fast the road was moving down the screen, which in turn depended on how many lines were scrolled at a time. Scrolling was a specialized form of byte manipulation.

The third module of the package accessed the game I/O controller of the Apple IIe and determined a value proportional to the position of the paddle. The values
ranged between 0 and 255. Based on the paddle value, the position of the car was incremented or decremented until the car's horizontal coordinate value coincides with the paddle value. The gain and the lag that were to be incorporated in the paddle input value was a part of this paddle routine. The fourth module of the package drew the shape of the car, or the shape of the object on the screen at the specified coordinate location based on the input value from the paddle.

The fifth module of the package checked whether the object was off the road or not. The object's left front corner was compared with the left and right edge of the road. If the object was inside the road, then the module produced an auditory sound that was created by accessing the speaker once for every iteration. On the other hand, if the object was off the road, the speaker was accessed rapidly 112 times per iteration to produce a distinctly different auditory sound to inform the subject that the object was off the road. The sixth module of the package collected the current input value from the paddle, the current left edge of the road, and the current left front corner of the object, then stored these values in RAM (Random Access Memory). After finishing the experiment, all experimental
data would be saved on floppy disks for statistical analysis.

Satish Tandon and Helmut T. Zwahlen (1987) developed another BASIC program for Apple IIe computer used to enhance the tracking task package mentioned previously. The new program had a main menu that allowed the experimenter to run the statistical program in order to analyze the experimental results or to run the psychomotor tracking task package designed by Viswanathan and Zwahlen (1986). The statistical program recalled the experimental data from the floppy disks and generated a statistical report including the lateral position average value, the standard deviation, and the accumulated percentage of the object positions.

1.4 CONSIDERABLE EXPERIMENTAL FACTORS

Since driving a vehicle on a highway was the most popular case of the preview tracking tasks in the real world, it was the best case to study for understanding the importance of preview tracking tasks. There were no other reasons that did not select vehicle driving as the main theme of the simulator that would be built in this thesis study.

Some experimental factors which would influence the man's tracking performance obviously had been researched by
a number of investigators and discussed in the above sections, like time lag, control gain, and preview time. Appropriate control aid (gain) and preview time could make up for the influence of time lag. Object speed (rate of movement), road width and number of curves would be also effect man's tracking performance. The larger the road width, the easier the vehicle control. The higher the object speed and the number of curves, the more difficult the car driving. These six factors were integrated into the simulator system to demonstrate different appearances of preview tracking tasks to students in class. Figure 1.2 illustrated the relationship between these six factors and preview tracking performance.

In conclusion, an integrated package which performed both the preview tracking experiment and the statistical analysis on an IBM Personal Computer (PC) or an one hundred percent compatible computer was developed. Under the advanced technology of both graphical software and microcomputer hardware, improvements in the package's practicability and user friendliness were foreseeable.
Figure 1.2 The relationship between preview tracking performance and some important experimental factors
2.1 OBJECTIVE

As mentioned earlier, the objective of this thesis project was to design a user friendly software package for measuring the human performance in preview tracking tasks executed on an IBM PC. The package was an effective and low-cost experimental tool to demonstrate a number of different aspects of human tracking performance to a human factors engineering class, and to allow the human tracking experiments to be conducted in laboratories also.

2.2 BASIC CONCEPTS

1. For the convenience of judging human tracking performance, the package should be able to save the data from the experiment on disks or other storage devices in order to analyze the data at a later time. The package could also output a complete statistical analysis report. The data collected in the experiment was (I) lateral the position of the object, (II) the position of the left road edge, and (III) the speed of the object. From (I) and (II), the distance between the object and the center
line of the road could be easily identified. Besides, (I) and (III) could also demonstrate the change of the input value from the input devices. The above mentioned three types of data could help experimenters to realize how subjects reacted to different experimental conditions. In addition, the output report should show the average value of the object lateral position, the standard deviation of the object lateral position, the accumulated object position percentage, the total time and distance that the object was off the road, as well as the average object speed, etc. Histogram charts were also provided in this report.

2. The experimenter could set up or change the value of the experimental variables at the beginning of the experiment. These variables were: object shape, object speed, road width, relative frequency of curves on the road, time lag or time delay, gain of the control input, preview time or preview distance, experiment time, subject adaptation time, and initial seed of the random number generator. The subject could only control the lateral position of the object using the yoke. If the object speed was set up as a variable by the experimenter in
the beginning, then the subject could control the object speed by using the pedals (accelerate, brake).

3. Instant and accurate reaction of the object to the changing input by the subject was needed, so that the subject could react in a timely fashion to the input and make the corrective movements.

4. The system display simulated the real situation as closely as possible. The scene was shown in a top-view perspective. The width of cars could not exceed 8 feet, which was the maximum car width allowed by the US government.

5. During the tracking task, a noticeable sound was produced when the car was off the road.

2.3 REQUIREMENTS

2.3.1 Hardware requirements

The programs were designed to run under the MS Windows operating environment. The hardware requirements of this project were similar to the hardware requirements specified for the MS Windows 3.X in enhanced mode.

1. A true Intel 80386/80486 CPU based IBM PC compatible computer was required. Although Intel 80286 CPU based PCs could run MS Windows 3.1 in standard mode, they were not suitable for the project because of
their low performance. A mathematical coprocessor was recommended to improve system performance if an 80386 CPU based computer was used.

2. The computer must be equipped with a VGA, a VGA-Mono, or a better Windows supported video adapter. A Windows acceleration card with 1 MB VRAM was strongly recommended, to raise the graphics-drawing speed. A VGA, a VGA-Mono, or a better monitor which could display pictures in a 640 pixels * 480 pixels resolution with 16 colors (256 colors monitor recommended) was also needed.

3. A hard disk with at least 2 MB free space had to be provided in the computer. The Programs should be installed onto the hard drive. The experimental data could be stored on either the hard disk or on the floppy disks after the experiment.

4. MS Windows requires at least 2 MB extended RAM to run in the 386 enhanced mode. Additional RAM was strongly recommended to improve the system performance.

5. A mouse that was supported by MS Windows was required.

6. A MAXX II Yoke, MAXX Pedals, or compatible products were also needed for the subjects to control the
lateral position of the object on the screen. The yoke was preferred over a proportional joystick because it resembles the automobile steering wheel. The subject had to rotate the yoke in order to produce the lateral movement of the selected object (a car or a circle). The MAXX Pedals could be used to simulate the gas pedal and the brake pedal on a real car. The MAXX Pedals and MAXX II Yoke should be interfaced with an appropriate interface card (game card).

7. A sound board with a Windows supported sound drive was helpful, because it could generate the proper audio effect to warn subjects when the car or the circle was off the road.

8. A Windows supported printer was required for printing the final reports.

2.3.2 Software requirements

1. MS Windows version 3.X was absolutely required. In order to run MS Windows, the MS DOS version 2.1 or higher should have been set up in the computer.

2. To take full advantage of the easy-to-use windows graphical interface, the programs were true Windows applications with a well-designed user interface.
3. The programs designed for the project must be fast enough to draw pictures and to collect data in the real time. Most PCs could run programs to animate objects at 4-14 frames per second (fps) (Microsoft Press, 1991). In order to provide quality animation images, 14 fps was the expected animation refresh rate in this project.

4. The experimenter must be able to choose the options or change variable values from the dialog boxes before running the experiment.

5. The programs were able to print out the detailed analysis report with the histogram charts, the average lateral position value, the standard deviation, and any other data that could help experimenters to understand the preview tracking behavior and performance.

6. A Windows application integrated development package was required to develop the programs. This package should have a resource code editor to edit .RC or .RES files. These files contain all the necessary information for the menu bar, dialog box, cursor, icon, and other kinds of resources used in the programs.
7. For some specific hardware devices, (yoke, VGA card, etc.) the related software device drivers like the game card driver and the video adapter card driver, had to be installed and loaded before running the programs.

2.3.3 Ergonomic requirements

1. The Yoke used must be easy to hold and to rotate. A left rotation performed by the subject should result in a proportional lateral left shift of the object on the screen, and a right rotation should result in a proportional lateral right shift of the object.

2. The pedal used must be sensitive enough to adjust the object speed easily. The right-side pedal was used to simulate the accelerator pedal in a real car and the left-side pedal served as a car foot brake.

2.3.4 Experimenter requirements

1. The experimenter was not required to have experience and knowledge in programming, but basic knowledge about operating MS Windows programs would be helpful for executing the experiment.
2.3.5 Subject requirements

1. The subject does not need any experience in using computers and MS Windows.
2. The subject should be able to respond to the systems by both auditory and visual stimuli.

2.3.6 System Requirements

A preview tracking task was presented in this thesis project. A top-view of a two-lane road with tangent and curve sections scrolling from the top of the screen to the bottom creates the car moving effect. In addition, a broken center line was displayed on the road to emphasize the driving and speed effect. The driving object used in this package could be either a circle or a car. The curves generated by the program must be fairly smooth. The refresh rate of animation must be fast enough to avoid major display of scene flickering.

The software developed for this project was considered a user friendly package requiring no programming knowledge for both the experimenter and the subjects. A series of dialog boxes for the experimenter to set up the experimental variables were given in the beginning of the experiment.
During the experiment, data about the location of the left edge of the road, the front left corner of the object, and the object speed were collected at each sampling interval and stored in RAM. The experimental data recorded would be transferred from RAM to either the hard diskette or the floppy diskette after each experiment. The experimental results could be analyzed and completely recalled to generate a detailed report. Histograms of the experimental data could be produced.

2.4 CONSTRAINTS

1. The major constraint in this project was the use of MS Windows version 3.0 or higher. Although MS Windows had a well designed graphical user interface which allowed for ease of development of user friendly application programs; the executing speed of such Windows applications under the Windows environment was generally slower than that of general DOS applications. In particular, this was true for graphical screen output. Therefore, it was extremely difficult to match the software requirement of drawing pictures at 14 frames per second (fps). Using MS Windows, a system performance tradeoff was unavoidable.
2. Because of the choice of Windows as the operating environment in this project, the amount of RAM was a strict constraint in developing this program as well.

All pictures displayed on the computer screen required a large amount of random access memory (RAM). The larger the picture, the more memory required to store it. If there was insufficient memory available, MS Windows would create a swap file on the hard disk as virtual memory that increases the total amount of memory. Because of the frequent disk access, virtual memory tended to be very slow. This resulted in a slow graphical output and slow program execution. In order to maintain high system performance, the animation zone (size of animation pictures) should be restricted such that all graphical activities may be performed from the installed RAM.
There were two main tasks assigned to the project. One was to develop a program to run preview tracking tasks on a microcomputer and the other was to design a program to analyze the data generated during the experiment using the above preview tracking package. In order to simplify programming, to reduce the program developing time and to avoid conflicts resulting from different system operation functions, the project was divided into two separate programs. The first program, SIMULATOR, was designed to run the preview tracking tasks on a PC. The second one, TRACER, was designed to analyze the experimental results obtained from SIMULATOR. Figure 3.1 illustrated the function architecture of the software package.

Generally, a C language program was constructed by some functions. The constituent functions in the C source code files(.C) of SIMULATOR and TRACER were listed below.

1. The functions of SIMULATOR were:
   a. WinMain,
   b. MainWndProc,
   c. SimmWndPrco, CRWndProc,
Figure 3.1 Function Architecture of the whole package
d. AboutProc, Dial0Proc, Dial1Proc, Dial2Proc, Dial3Proc, Dial4Proc, FPDProc, FPTProc,
    e. InitialRandomNumber, DrawRoad, CollectData, YokeInput, CarPosition, SetXValue, and SelectShape.

The WinMain and MainWndProc were the procedures to register, create, and maintain the SIMULATOR's main window. The primary function of SimmWndProc was to create and control the child window. Child window is window that could be only displayed within the client area of the parent window. The CRWndproc was responsible for displaying the initial window with copyright information. The dialog box procedures were listed under item d. Procedures listed under item e were created to perform the specific repetitive jobs.

2. The functions of TRACER were:
   a. WinMain,
   b. FrameWndProc,
   c. TextWndProc, GraphWndProc, InfoWndProc, RstWndProc, CRWndProc,
TRACER was a Windows application with a Multiple Document Interface (MDI); therefore, its programming structure was not the same as the structure of SIMULATOR. The WinMain and FrameWndProc were the functions to register and handle the main window and one MDI client window. TextWndProc, GraphWndProc, InfoWndProc, and RstWndProc were child window control procedures. CRWndProc had the same duty as it was assigned in SIMULATOR. The functions listed under d. were dialog box process procedures, and the functions listed under e. were the specific function procedures.

In addition to the C source code files (.C), some special files, which could not be found in normal DOS applications, were also necessary to create a Windows application. The additional files were the resource-definition file (.RC) and the module definition file (.DEF). These two files defined all system resources and information that were needed for compiling and linking Windows programs.

3.1 SYSTEM OPERATION FUNCTIONS OF SIMULATOR

The WinMain function was the entry point of all Windows applications like the main function found in standard C language applications. WinMain initialized and registered the window class (a window was always created based on a
window class), then called MainWndProc to create a main window by calling CreateWindow function. When the main window was fully visible on the video display, WinMain was then ready to read input from the operator. It creates a message loop using a while statement to retrieve messages from the Windows message queue and to dispatch these messages to the proper window functions until it receives the WM_QUIT message informing WinMain to jump out of the message loop and terminate SIMULATOR.

MainWndProc was the main window procedure of SIMULATOR. The primary duty of MainWndProc was to decide what the main window displays in its client area and how the window responded to user input by processing different types of message. Figure 3.2 Showed the flowchart of MainWndProc.

Basically, the kernel of MainWndProc was a switch-case statement. Every window procedure described in this chapter had at least one switch-case statement to process Windows messages. The SIMULATOR's main window procedure used the switch-case structure to manage three common messages, WM_CREATE, WM_DESTROY, and WM_COMMAND, and determined how to process them accordingly. The other messages were automatically passed to DefWindowProc, a predefined function for default processing of messages in Windows.
Figure 3.2 Flowchart of MainWndProc
Figure 3.2 Flowchart of MainWndProc (Continued)
Figure 3.2 Flowchart of MainWndProc (Continued)

(*) After preview is set up, initial speed and object speed can be set.
In the WM_CREATE case, the seven initial data (.INI) files, SETUP.INI, LB.INI, RB.INI, LD.INI, RD.INI, LU.INI, and LD.INI, which stored numerical data of curve shapes, were retrieved by the SIMULATOR. Figure 3.3 showed the related curve shapes of these .INI files, excepted for SETUP.INI file, which just stored the radius of the curves. The WM_DESTROY message was a messenger from Windows to notify MainWndProc that the main window was being destroyed. In the WM_DESTROY case, the PostQuitmessage function was called to post a WM_QUIT message in Windows application queue. WM_COMMAND was an input message indicates the operator's choice from the menu or the dialog boxes. Two pop-up menu items were listed on the main menu of SIMULATOR which were "OPTIONS" and "ABOUT". "OPTIONS" contained two sub-menu items, "NEW SUBJECT..." and "EXIT". Another sub-menu item "ABOUT..." was placed under the pop-up menu item "ABOUT". If the experimenter chose "NEW SUBJECT..." (whose ID was IDM_NEW) from the menu of SIMULATOR, a series of dialog boxes were prompted on the screen to ask the experimenter to initialize experimental variables. Then a new child window was created based on the window class "NewClass". If the menu item "Exit" (whose ID was IDM_EXIT) was selected by the user, the DestroyWindow function would
Figure 3.3 Related curve shapes of *.INI files
be called to inform Windows to destroy the main window of SIMULATOR. If the menu item "About..." was activated (in the IDM_ABOUT case), a function was prepared to show the dialog box "AboutBox" on the screen. ("..." meant at least one dialog box would be presented after the menu item was selected.)

SimmWndProc, the most important window procedure in SIMULATOR was called, when a new child window was being generated. Like other window procedures, SimmWndProc was also designed to process messages from Windows by a switch-case statement. Figure 3.4 illustrated the flowchart of SimmWndProc.

There were four messages that were processed in this window function, WM_CREATE, WM_TIMER, WM_PAINT, and WM_DESTROY. Since WM_CREATE was the first message that the window procedure could receive, it was a good place to initialize system global variables and local variables. In the case of WM_CREATE, most variable values specified by the experimenter were recalculated and prepared for specific formulas.

WM_TIMER was a Windows message to manage up to thirty-two timer inputs in Windows concurrently. Three timer inputs were set to perform different functions in SIMULATOR. Timer 1 with a five hundred millisecond timer interval was
Figure 3.4 Flowchart of SimmWndProc

(called by Windows or WinMain())

message

fault (other messages)

WM_CREATE

WM_TIMER

WM_PAINT

WM_DESTROY

A

C

E

G

B

D

F

H

( return (NULL) )

DefWindowProc()
Figure 3.4 Flowchart of SimmWndProc (Continued)
A

Proceeding experimental variables (lag, experiment time and subject adapt time)

1. Set title bar, 2. set subwindow "WindShield", 3. disable menu items

<load object shape>

get object size and load bitmap "CARBITMAP"

get object size and load bitmap "CIRCLEBITMAP"

proceeding experiment variables (curve rate, roadwidth, and random)

set nLR, nCX, nRX, and nLX

nIndex

409 > nIndex > 0

set X-coordinate position of center line

nIndex = nIndex - 1

B

E

get current device controller (screen)

got preview distance

display object bitmap

create pen and brush

draw road on the screen

delete pen and brush

release device controller

collect experiment data

validate client screen area

Figure 3.4 Flowchart of SimmWindProc (Continued)
Figure 3.4 Flowchart of SimmWndProc (Continued)
created to initialize the client area on the screen of the child window. This initialization included things like drawing sub-window frames and displaying the graduated scale of the "SpeedBar" which was used to inform the subject about the object speed. The timer interval of timer 2 was also five hundred milliseconds. When SIMULATOR received the WM_TIMER message with timer ID 2 from Windows, the two sub-windows, "SpeedBar" and "BrakeBlock," would be repainted and a beep would be generated if the object was off the road. The last timer, timer 3, with a timer interval of 100 milliseconds was used to notify SIMULATOR to update the contents in the sub-window "WindShield", and thus established a continuous animation in 10 Hz frequency.

WM_PAINT was a message sent by the Windows kernel if Windows or the applications request the partial updating of an application window. The primary function of the WM_PAINT process was to draw the road and to display an object (a car or a circle) controlled by the subject. In the WM_DESTROY case, a routine to save the experimental data in specific formats was available. When the child window was destroyed, a dialog box then presented the subject with the option to save or discard the experimental data.

CRWndProc was a short procedure in SIMULATOR. It was used to create an initial window for loading the bitmap file
CR.BMP, to the screen. All detailed description of this package (including the copyright information) had been pre-drawn on this bitmap file. After showing the bitmap file for six seconds, CRWndProc would receive a WM_TIMER message with timer ID 21 from Windows kernel to inform that the child window should be close. Figure 3.5 showed the initial window with the bitmap file, CR.BMP.

The dialog boxes, "About", "Dial0", "Dial1", "Dial2", "Dial3", "Dial4", "FDP", and "FDT", used in SIMULATOR were designed to exchange information with the experimenter. Each dialog box had a template which defines the features and the contents of the dialog box in the resource script file (.RC). Every dialog box procedure could create the related dialog box to process necessary Windows messages.

WM_INITDIALOG was always the first message sent to the dialog box procedure. Like the WM_CREATE message in the window function, default values of the experimental variables were initialized while Windows was processing the WM_INITDIALOG message. WM_COMMAND was also used in dialog box functions. Most child-window-controls (like scroll bar, check box, list box, and so on) sent this message to their parent window indicating that the incorporated value had been set or revised. If the "OK" button located in every dialog box was pushed, all setup values of experimental
Figure 3.5 The initial window used in SIMULATOR and TRACER
variables were accepted by SIMULATOR. Otherwise, all input values were lost.

Finally, there were several procedures that were developed for specific repeated tasks. They were listed and described below.

1. *InitialRandomNumber*: to generate a random number
2. *DrawRoad*: to draw the road on the screen
3. *CollectData*: to collect the experimental data
4. *YokeInput*: to receive input control value from the yoke and the pedals
5. *CarPosition*: to distinguish the lateral position of the object
6. *SetXValue*: to set the X-coordinate position of the left road edge
7. *SelectShape*: to select the curve shape of the road.

3.2 SYSTEM OPERATION FUNCTIONS OF TRACER

Except for the added Multiple Document Interface (MDI), the system operation procedure of TRACER was simpler than the one of SIMULATOR. The *WinMain* function in TRACER was similar to that in SIMULATOR and its primary function was to register the window classes and dispatch messages from the Windows kernel to the other window procedures. Since a set
of accelerator keys was defined in TRACER for the convenience of system operation, the LoadAccelerators function had to be added to WinMain to load the accelerator table into the memory and get the accelerator table's handle. Besides LoadAccelerators, the other two functions, TranslateMDISysAccel and TranslateAccelerator, had to be set into the message loop and translate the keyboard messages.

In FrameWndProc, the main window procedure of TRACER, the same messages that occurred in MainWndProc of SIMULATOR were used again. Because the MDI interface was added, some message process routines were changed. Figure 3.6 showed the flowchart of FrameWndProc.

In WM_CREATE, a virtual MDI client window was created with the window class "MDICLIENT", which was a preregistered window class of MDI. In WM_COMMAND, four menu items, "Cascade", "Tile", "Arrange Icon", and "Close all" were added to help the operator to manage various child windows. These were pre-defined system functions in Windows. Menu items, such as "Print", which were not handled by the FrameWndProc, were transferred to other child window procedures by the function SendMessage.

In TRACER, CRWndProc was asked to show the initial window on the screen like it did in SIMULATOR. After showing the initial window for six seconds, Windows would
Figure 3.6 Flowchart of FrameWndProc
Figure 3.6 Flowchart of FrameWndProc (Continued)
open dialog box
5

input .RST file
name

yes

read data from
 .RST file

no

set title bar

create a child
window

send message
WM_CLOSE to
the child window

send message
WM_MDITILE to
MDI client window

send message
WM_MDITILE to
MDI client window

send message
WM_MDICASCADE to
MDI client window

send message
WM_MDIICONARRANGE to
MDI client window

find active
child window

send message
WM_CLOSE to
the child window

Figure 3.6 Flowchart of FrameWndProc (Continued)
ask CRWndProc to close the initial window. The main job of the other four child window procedures was to output the content of the data files, .DAT, .INF and .RST, to the assigned output devices after reformatted or analyzed. TextWndProc and GraphWndProc were designed to retrieve the experimental data from each .DAT file. The former displayed the data in tabular form, and the latter illustrated the data graphically. InfoWndProc was the window procedure to create a child window for the operator to review all experimental variables (logged in .INF files) at any time. RstWndProc analyzed the experimental results which were recorded in .RST files and built a complete analysis report for each experiment. The two possible output devices were the monitor and the printer. The data output functions for these two devices were very similar.

Several dialog boxes were used in TRACER for the same purpose as in SIMULATOR. The "Abort Printing" dialog box was different from the other dialog boxes, because it was modelless while all others were modal dialog boxes. The difference between these two types of dialog boxes was that the modelless dialog box allowed users to work in the parent window, while the modal dialog box did not. Therefore, it was recommended to use the modelless dialog box to manage printing jobs.
Several functions were defined in TRACER for specific repeated tasks. They were listed and described below.

1. CloseEnumProc: to find and close all opened and non-active child windows on the screen
2. GetValue: to read data from data files
3. GetPrinterDC: to get the device driver of the current printer set in Windows.

3.3 RESOURCE DEFINITION FILES

Most Windows applications used a variety of resources, such as menus, bitmaps, icons, and dialog boxes, each defined in a file called the resource-definition (.RC) or resource script file. These resources could be edited by any resource editor which come with the complete program development packages. SIMULATOR has a resource-definition file named SIMM.RC. TRACE.RC was the resource-definition file of TRACER.

Both SIMM.RC and TRACE.RC contain several ASCII representations of resources and also refer to other files (either ASCII or binary files such as .BMP, .ICO, etc.) that contain resources. The detailed description of both resource script files were listed in Appendix C and D.
3.4 MODULE DEFINITION FILES

The file with the extension name .DEF was a required file to build a Windows application. It was called the "module definition file", which recorded the characters of the program code and data segments, the size of the local data heap, and the stack size of the program.

In SIMM.DEF and TRACE.DEF, both code and data segments were set to PRELOAD and MOVEABLE. These two keywords were the normal options for Windows applications. PRELOAD and MOVEABLE meant that Windows system could immediately load code and data segments into memory when needed. If the other Windows applications needed an entire block of memory which had been occupied by these two segments, Windows might move these two segments (code and data segments) to other places in memory. The code segment also has the parameter DISCARDABLE which specified that the code segment could be removed from memory by Windows if free memory space was scarce. The keyword MULTIPLE, the last parameter of the data segment, specified the application restart, when a new data segment was created.

The HEAPSIZE was used to specify the size of extra local memory which was available for data allocating. Some of the local memory was used to keep the child windows' data in switching focus among those child windows in TRACER.
Therefore, the heap size of TRACER was a little bigger than the size of the SIMULATOR's local heap.

The STACKSIZE specifies the size of the stack. Each Windows application had its own stack. At least 4096 bytes were required for a small Windows program. Since several non-static variables were used in both SIMULATOR and TRACER, they needed bigger stacks.
Programming a Windows application in C language was quite different from programming a DOS application. Almost all of the general purpose function libraries available for the C language did not apply to Windows applications, especially the function libraries concerned with file I/O and memory management. The most important system functions used for the Windows applications had been defined in windows.h. Except for those programmer defined functions, all functions described in this chapter were defined in windows.h.

4.1 SCREEN DISPLAY OF SIMULATOR

There were two factors that should be considered before a screen layout could be designed. First, the subject had to be able to retrieve enough information from the screen in real time. Second, the system should not spend too much time on drawing the screen pictures that would influence the other system functions’ executing speed.

The display resolution of a normal VGA monitor was 640 pixels x 480 pixels with at least 16 different colors. Disregarding the height of the application title bar and the
main menu bar, the final size of the customer client area was no more than 640 pixels x 440 pixels.

There were three sub-windows to display the experimental information in the client area: "WindShield", "SpeedBar", and "BrakeBlock". The "WindShield" was the largest among the three sub-windows. Its width and height were 500 pixels and 410 pixels respectively. The origin of this sub-window was at an absolute location of (10, 10). Figure 4.1 showed the coordinate system and mapping mode, MM_TEXT mode, which was the coordinate system used in the programs only. In this package, 4 pixels represented 1 foot in the real world (no matter in vertical or in horizontal). The "WindShield" was created to display the animation pictures on the screen, and its content could be updated at a frequency of 10 Hz.

The "SpeedBar" and the "BrakeBlock" were positioned at the right hand side of the client area. The former was in charge of informing the subject about the speed of the object, while the latter was in charge of displaying information about the braking intensity. If the subject depressed the brake pedal, the color of the "BrakeBlock" would be changed from bright blue to red. SIMULATOR refreshed the present speed of the object shown in the "SpeedBar" in 0.5 second intervals, while the "BrakeBlock"
Figure 4.1 The coordinate system used in developing the package
was refreshed when the brake pedal was either depressed or released. Figure 4.2 showed the whole screen design of SIMULATOR.

4.2 ALGORITHM OF ROAD GENERATION

The road generation algorithm was the core of the SIMULATOR. Using an unsuitable algorithm would result in inadequate animation pictures on the screen. In order to develop a well-designed algorithm, several ideas and suggestions had been discussed in class with the instructor and classmates to improve the efficiency of the algorithm. Table 4.1 listed the ideas and suggestions that had been discussed in class and tested in SIMULATOR. Their disadvantages and advantages were also listed on the same table. Finally, an integrated algorithm had been developed successfully for SIMULATOR. This algorithm drew the road in real time, and it took the advantage of the pre-drawn curve shapes.

Two different functions in SIMULATOR were responsible for the road generation. The first function was called `MainWndProc`. This function retrieved data from few specific initial data (.INI) files. The following .INI files, `LD.INI`, `RD.INI`, `LU.INI`, `RU.INI`, `RB.INI`, and `LB.INI`, contain the X-coordinates of every point within the curves of different
Figure 4.2 The screen design of SIMULATOR
<table>
<thead>
<tr>
<th>Suggestions and Ideas</th>
<th>Disadvantages</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using generated random numbers to determine the curve length and the curve shape.</td>
<td>(1) The generated curve rate is not the rate required by the experimenter. (2) The curve shapes and sizes were not in control.</td>
<td>More easily coding the program.</td>
</tr>
<tr>
<td>Changing the background color, then drawing the road in white.</td>
<td>(1) More time is required to redraw the client area. (2) The quality of the animation pictures is poor.</td>
<td>No obvious advantage</td>
</tr>
<tr>
<td>Using pre-drawn bitmaps to construct the animation.</td>
<td>(1) More disk space and memory were needed to store bitmaps.</td>
<td>Higher animation refreshing speed. The speed is depending on how big bitmaps were.</td>
</tr>
<tr>
<td>Using the broken lines to replace the solid lines to represent the road borders.</td>
<td>(1) Not much improvement in raising the animation refreshing rate. (2) It is difficult for the subject to identify the road edges.</td>
<td>No obvious advantage</td>
</tr>
<tr>
<td>Using a better video adapter card or a better PC.</td>
<td>It is expensive.</td>
<td>It is the easiest way to raise the system performance.</td>
</tr>
<tr>
<td>Using supported function libraries to draw the curves.</td>
<td>Windows function libraries do not support these functions.</td>
<td>There were some third-party's customized function libraries in the market. Using these libraries may simplify the programming problems.</td>
</tr>
</tbody>
</table>
shapes (see Figure 3.3). Another .INI file, SETUP.INI, contained the radius of the curves, 76 pixels (19 feet). The road displayed on the screen consists of small tangent and curve sections with a length of 200 pixels (50 feet) per each section. By keeping the source code and the data in separate files eased the maintenance of software packages. If the experimenter wanted to change the shape of curves, he/she needed to understand the data files only. In this package, the experimenter did not need to understand the source code of the program.

After MainWndProc received the message WM_CREATE from the WinMain function, a subroutine in the WM_CREATE searched the seven .INI files in SIMULATOR's own working directory. If these .INI files were found, their contents would be retrieved and stored in the pre-defined data array NCI[]. Otherwise, the content of NCI[] was initialized to zero and the road shown on the screen would be a straight road.

The second function related to the road generation was called from SimmWndProc - the major child window procedure. This function generated the road with tangent and curve sections.

Before explaining the algorithm of road generation, there were a few variables used in SimmWndProc which should be explained in advance. These variables were closely
related to the road generation algorithm. \( nRanIndexLast \), \( nRanIndexNow \) and \( nRanIndexNext \) were three variables storing the random numbers which were generated by the C language random number generator. The initial seed of the random number generator was set by the experimenter in the beginning of the experiment. The generated random number was in the range of 0 and 255. The number was stored in \( nRanIndexNow \) and compared to \( fLineorCurve \), a float point number. The value of \( fLineorCurve \) depended on the relative frequency of the curves specified by the experimenter. If \( nRanIndexNow \) was greater than \( fLineorCurve \), then a tangent section (200 pixels) was generated. Otherwise, a curve section was generated. \( nRanIndexLast \) and \( nRanIndexNext \) were used to determine which types of curve should be drawn on the screen. After generating a new road section, the value of \( nRanIndexNow \) was sent to \( nRanIndexLast \) and the value of \( nRanIndexNext \) was transferred to \( nRanIndexNow \). A new number generated by the random number generator would fill up \( nRanIndexNext \), and the "comparing process" described previously would continue until the experiment ended.

The \( nLX \), \( nCx \), and \( nRX \) were three variables that identified the lateral position of the starting point of each new road section and the latest point of the last section. \( nCx \) was defined as the starting point of the first
section of the road. It was derived from the following formula.

\[ n_{CX} = 10 + \left( 500 - n_{RoadWidthPixel} \right) / 2; \]

In the formula, 10 was the X-axis position of the upper left corner of the sub-window "WindShield", and 500 was the width of "WindShield". \( n_{RoadWidthPixel} \) was the road width, which might be specified by the experimenter. \( n_{LX} \) was left 76 pixels to \( n_{CX} \) and \( n_{RX} \) was right the same distance to \( n_{CX} \). Each road section always ended and started in one of the three points. Both \( n_{LX} \) and \( n_{RX} \) were the virtual boundaries of the road left edge curves. Figure 4.3 showed the positions of \( n_{LX} \), \( n_{CX} \) and \( n_{RX} \). In order to emphasize positions of the three points, the solid lines of road borders were overlaid by the broken lines used to identify these three points on figure 4.3.

The \( n_{LR} \) was the variable that specified the curve direction. When its value was non-zero, the curve was drawn from right to left. If the value was zero, the curve was drawn from left to right. When the curves reached their left side or right side virtual boundary (\( n_{LX} \) or \( n_{RX} \)), the value of \( n_{LR} \) would be changed from zero to one or from one to zero.

The algorithm for the road generation was illustrated in Table 4.2. The table used all above mentioned variables.
Figure 4.3 the positions of nLX, nCX and nRX
Table 4.2 The algorithm of road generating

<table>
<thead>
<tr>
<th>Current Random Number</th>
<th>Last Section Shape</th>
<th>X-coordinate Position of Lastest Pixel</th>
<th>Curve Direction</th>
<th>Next Random Number</th>
<th>Road Shape of New Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>(nRanIndexNow)</td>
<td>(nLastShape)</td>
<td>(nFinalXPos)</td>
<td>(nLR)</td>
<td>(nRanIndexNEXT)</td>
<td></td>
</tr>
<tr>
<td>&gt; Curve Rate x 256</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Straight Line</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Straight Line</td>
<td>nCX</td>
<td>Left-&gt;Right (0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Straight Line</td>
<td>nCX</td>
<td>Right-&gt;Left (1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Straight Line</td>
<td>nLX</td>
<td></td>
<td></td>
<td>Curve(LB)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Straight Line</td>
<td>nLX</td>
<td></td>
<td></td>
<td>Curve(RB)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Straight Line</td>
<td>nRX</td>
<td></td>
<td></td>
<td>Curve(LB)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Straight Line</td>
<td>nRX</td>
<td></td>
<td></td>
<td>Curve(RB)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(LB)</td>
<td>nRX</td>
<td></td>
<td></td>
<td>Curve(RB)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(LB)</td>
<td>nRX</td>
<td>&lt;= Curve Rate x 256</td>
<td></td>
<td>Curve(RU)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(RB)</td>
<td>nLX</td>
<td></td>
<td></td>
<td>Curve(LB)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(RB)</td>
<td>nLX</td>
<td>&lt;= Curve Rate x 256</td>
<td></td>
<td>Curve(LU)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(LU)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(LD)</td>
<td></td>
<td>&gt; Curve Rate x 256</td>
<td></td>
<td>Curve(LB)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(LD)</td>
<td></td>
<td>&lt;= Curve Rate x 256</td>
<td></td>
<td>Curve(LU)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(RU)</td>
<td></td>
<td></td>
<td></td>
<td>Curve(LD)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(RD)</td>
<td></td>
<td>&gt; Curve Rate x 256</td>
<td></td>
<td>Curve(RB)</td>
</tr>
<tr>
<td>&lt;= Curve Rate x 256</td>
<td>Curve(RD)</td>
<td></td>
<td>&lt;= Curve Rate x 256</td>
<td></td>
<td>Curve(RU)</td>
</tr>
</tbody>
</table>

(*) If nRanIndexNext is greater than curv rate x 256, next section will be straight line. Otherwise, next section will be a curve.
to illustrate the logic of the algorithm. The algorithm was represented by using *if* or *switch-case* statements placed in the function called *SelectShape*. Figure 4.4 showed the flowchart of *SelectShape*. When a new road section was required by *SimmWndProc*, it called *SelectShape* to check the above mentioned variables in order to find the proper tangent or curve section gradually. For example, if *SelectShape* found that

(1) *nRanIndexNow* was smaller than the relative curve frequency * 256,
(2) the last road section was a tangent section,
(3) the 200th point of the last section was in *nLX*, and
(4) *nRanIndexNext* was also greater than the relative curve frequency * 256, \((1) \cap (2) \cap (3) \cap (4)\)

then the new road section should be the *LB* curve. The related curve shape was shown in Figure 3.3.

The left road edge was generated using the algorithm mentioned above. The right road edge was always derived from the left road edge with the same shape.

4.3 INPUT DEVICE CONTROL ROUTINE

A MAXX II Yoke and MAXX Pedals were used in the project to simulate the steering wheel, the gas pedal and the brake pedal. The MAXX hardware was compatible with the
called by
SetXValue

Find the X-axis
position of the latest
point and road

generate new
random number

nRanIndexNow

A
B
C
D
E
F
G
H
I
J
K
L
M
N

nLastShape

nNowShape: 6

nNowShape:
0 --> straight Line
1 --> LB curve
2 --> RB curve
3 --> LU curve
4 --> LD curve
5 --> RU curve
6 --> RD curve

(return 0)

Figure 4.4 Flowchart of SelectShape
Figure 4.4 Flowchart of SelectShape (Continue)
Figure 4.4 Flowchart of SelectShape (Continue)
commonly used joysticks. However, they were not standard input devices for Windows, and therefore not supported by the general function libraries. Only the special multimedia programming library had some pre-defined functions to help programmers to set these devices up and to receive input signals from them.

In order to improve the system efficiency and to decrease programming difficulty, a short 8086 assembly subroutine was placed into function YokeInput. The short assembly subroutine replaced the complicated pre-defined I/O functions. The IBM BIOS created after July 10 1986 supports joystick via interrupt 15h, function 8400h. It was a relatively simple task to receive data and signals from joysticks through the game ports by checking the value of specific registers in the CPU. The program sent two hex values, 84h and 01h, to the CPU's register AH and DX in every sampling iteration. After receiving these two values, the BIOS loaded several sets of data from the game ports and stored them in four CPU's registers (AX, BX, CX and DX). Then, YokeInput would call interrupt 15h to collect two numbers from register AH (a part of register AX) and CX.

The number obtained from register AH was the X-axis input value from the yoke, and the number obtained from
register CX represented the X-axis input value from the pedals. The yoke could take on the X-axis values from 6 to 68 with a neutral position of 37. If the input value from the yoke was less than 37, then the subject had made a right shift of the object's horizontal axis position. Otherwise, if the input value was greater than 37, a left shift of the object's position had been made. The X-axis input value of the pedals ranged from 5 to 50. If the subject depressed the right pedal (gas pedal), the input value obtained from the CX register ranged from 24 to 50 depending on the pedal position. The deeper the pedal was depressed, the larger the value. On the other hand, if the subject stepped on the left pedal (brake pedal), the input value obtained from the CX register ranges from 5 to 20. The deeper the pedal was depressed, the smaller the value.

The input gain and the time lag incorporated with the yoke input value would be a part of the function YokeInput. When the yoke input value obtained from the register AH was transferred to the variable nYokeBuffer, SIMULATOR would check the value of the variable nLags which determined the multiple of the time lag by one sampling iteration. If nLags was equal to zero (no time lag), the value of variable nYokeBuffer would be read soon. Otherwise, the value of nYokeBuffer would be sent to the defined data array nLag[].
The value would be retrieved again after nLags iterations to generate the effect of a time delay. The value of variable nXCarPosTemp was the object lateral position and it would be recalculated according to the following formula.

\[ nXCarPosTemp = nXCarPosTemp + [(nYokeInput - 37) \times nGains]; \]

The value of nYokeInput was transferred from nYokeBuffer or nLAG[]. The variable nGains represented the gain for the control input from the yoke. The new position of the object was equal to the original object's position plus the difference between the yoke input and the center position of the yoke times the gain for the input control.

If the experimenter decided to use variable speed control in the beginning of the experiment, the input value from the pedals affected the object speed. Otherwise, the object speed was always equal to the initial speed that the experimenter had selected during setup.

In the YokeInput function, there was an internal switch-case structure which determined the relationship between the object speed and the pedal input value. The value obtained from the CX register was sent to the variable nPedalInput. If the value of nPedalInput ranged from 20 to 50, it meant that the subject had stepped on the right pedal (gas pedal) to change the object speed. The program
determined a theoretical object speed stored in the variable \textit{nIdeadSpeed} by checking the internal switch-case structure. If the value of \textit{nIdeaSpeed} was different from the value of \textit{nSpeedBuff} that represented the current object speed, \textit{nSpeedBuff} was incremented or decremented by one in every sampling iteration until \textit{nSpeedBuff} reached the same value as \textit{nIdeaSpeed}.

If the value of \textit{nPedalInput} was smaller than 16, the subject had stepped on the left pedal (brake pedal) to slow down the object. The value of the \textit{nSpeedBuff} would be decreased by three in every sampling iteration. But the value of \textit{nSpeedBuff} could not be less than zero or greater than 120. The flowchart of \textit{YokeInput} was illustrated in Figure 4.5.

4.4 ERASE AND RE-DRAW ROUTINE

The key of the animation was to erase and re-draw the pictures at a high frequency. In this project, \textsc{Simulator} refreshed the road shape ten times per second (10 Hz). \textit{SetTimer}, a Windows system function, was responsible for allocating the timers in the \textsc{Simulator} to perform the specific functions. There were 3 timers used in \textsc{Simulator} (see Section 3.1) and only timer 3 with a 100
called by SimmWndPro

get input value from CPU's registers, AX and

Is time lag > 0?

yes

the input value from yoke is put in nLag[], it will be retrieved after nLags seconds to generate the effect of time lag

no

Is the value of nYokeInput over limitations?

yes

nYokeInput = nYokeMax

no

nYokeInput = nYokeMin

get X-coordinate position of object

Is the object position over limitations?

too small

nXCarPosTemp = 11

too large

nXCarPosTemp = 510 - nobjectWidth

Find a theoretical speed of input value from pedals, and nBrake = 0

if the current speed (nSpeedBuff) >

yes

no

nSpeedBuff--

If nSpeedBuff < the theoretical speed

yes

no

nSpeedBuff++

A

B

Figure 4.5 Flowchart of YokeInput
If nBrake = 1

\( n \text{SpeedBuff} = \text{Max. allowed} \)

nSpeedBuff = 0

Figure 4.5 Flowchart of YokeInput (Continue)
millisecond timer interval was related to the animation of
the tracking tasks.

When the function SimmWndProc in SIMULATOR received the
WM_TIMER message from Windows and the timer ID was 3 (Timer
ID was set in wParam of message WM_TIMER), it checked the
value of the variable nIteration. If the value of
nIteration was not greater than the value of nRunTimes that
showed the length of the experiment time, SIMULATOR would
call function YokeInput to get the input value from the yoke
and pedals immediately. Then, SIMULATOR checked the
object's speed in order to decide how many pixels (the
variable nScrollPixels) should be scrolled down to generate
the car moving effect. Higher object speeds resulted in
more pixels to be scrolled down. To scroll the screen down,
the X-coordinate position of the left road edge of the nth
line, the value of XDisplayPos[n], was shifted to
nXDisplayPos[n+nScrollPixels], the value of
nXDisplayPos[n+1] was shifted to
nXDisplayPos[n+1+nScrollPixels], and so on. Then the
function SetXValue was called to generate the new data to
fill up the top nScrollPixels array cells whose contents had
been shifted to the other cells. After the data array
nXDisplayPixels[] had been filled out, a command
InvalidateRect was exported to Windows to erase the entire
client area of the sub-window "WindShield", then Windows would return a message WM_PAINT to SIMULATOR to notify that SIMULATOR can start to repaint the client area of the "WindShield".

Whenever SIMULATOR received the message WM_PAINT from the kernel of Windows, the function BitBlt was called to display the selected object bitmap (a car or a circle) at the proper position on the screen. Then the system function SetPixel was called by SIMULATOR to draw the road shape pixel by pixel. In SIMULATOR, the array nXDisplayPos[] stored the X-coordinate positions of the left road edge. The X-coordinates for the right road edge could be computed by adding the road width to the X-coordinates of the left road edge.

If the experimenter requests SIMULATOR to draw a center line on the road, the same method was applied. The position of the center line could be calculated by adding 1/2 of the road width to the X-coordinates of the left road edge. When the road was refreshed, the function ValidateRect was sent to Windows to maintain the present client area of sub-window "WindShield" until Windows received the InvalidateRect command again to start a new redrawing cycle.
4.5 OFF ROAD CHECKING

The value of the variable \texttt{nCarPos[nIteration]} and \texttt{nLeftPos[nIteration]} represented the X-coordinates of the object's left front corner and the X-coordinates of the left road edge. If the value of the \texttt{nCarPos[nIteration]} was less than the value of the \texttt{nLeftPos[nIteration]} (the left road edge) or greater than the value of the \texttt{nLeftPos[nIteration]} plus \texttt{nRoadWidthPixel} (the right road edge), the object was off the road, and SIMULATOR would send out a beep by calling the \texttt{MessageBeep} function. The sound effect of \texttt{MessageBeep} could be altered using the Control Panel of the Windows desktop.

In the project, SIMULATOR checked the object position only every 0.5 second in order to avoid overloading the Windows system. This was a necessary tradeoff between rapid checking responses and high animation refresh rates.

4.6 STORING EXPERIMENTAL DATA

Three types of data were collected by the system:
1. Current left road edge location at the preview distance
2. Current object position
3. Object speed.
During the experiment, SIMULATOR checked and collected data using 3 defined variables:

1. nXDisplayPos(position of the left road edge)
2. nXCarPosTemp(object position)
3. nSpeedBuff(object speed).

These three variables were stored in three large data arrays, nLeftRoad[], nCarPos[], and nCarSpeed[] in every sampling iteration. At the most, these arrays could hold three thousand sets of data for an experimental run of five minutes at the sampling rate of 10 Hz. After the experiment, a dialog box appeared on the screen asking the experimenter to specify the data file name. After the experimenter entered the file name and clicked the "OK" button, the program generated three files with the extensions .DAT, .INF, and .RST. For example, the file name entered by the experimenter was test1, then the three files, test1.dat, test1.inf, and test1.rst would be automatically generated by SIMULATOR. Test1.dat was used to store up to three thousand sets of experimental data transferred from the three data arrays, nLeftRoad[], nCarPos[], and nCarSpeed[]. Test1.inf was used to save all values of experimenter-controlled variables. Test1.rst was created to store the total number of object(car or circle) positions, the cumulative time that object moved off the road, and the
cumulative object off road distance. If the experimenter clicked "CANCEL" button, the dialog box procedure would return FALSE to the child window procedure SimmWndProc and all experimental data was discarded.

4.7 DATA PROTECTION IN MULTIPLE DOCUMENT INTERFACE (MDI)

The Multiple Document Interface (MDI) in TRACER was a user-interface standard for presenting and manipulating multiple documents within a single window. Every document was seen as a child window and had its own data content. This data content should be kept in memory space until the child window (the document) was destroyed.

There were three types of global data structures defined in the beginning of TRACE.C. They were defined to hold the data transformed from the .DAT, .INF and .RST files when TRACER was executing. When a child window was created, the function LocalAlloc was called to create a local heap for allocating the contents of the child window. If data assigned to these local heap was recalled by the child window procedure, the function LocalLock was used to lock the local heap before the data was retrieved from the heap or written into the heap. This function might protect the child window's data from disturbing other Windows applications and Windows functions. Then, LocalUnlock was
used to unlock the local heap after data had been retrieved or written.

When the message WM_DESTROY was sent to the child window procedure from the Windows system to notify that the child window was being destroyed, the local heap was released by calling the function LocalFree.

4.8 PRINTING

Printing from Windows was easier than from DOS. Windows had a graphical interface that matches the requirement of WYSIWYG (What You See Is What You Get) easily and all printer device drivers and functions have been standardized such that programmers did not need to worry about the problems resulted from printer differences.

The subroutine GetPrinterDC in TRACER was designed to get the current printer device driver information that was recorded in the file WIN.INI. When a user pointed the menu item "Print", the active child window would look for the current printer by calling subroutine GetPrinterDC. If the printer driver was found, a virtual memory device was created to simulate the printer device. The active child window would paint its contents on the virtual device as it does on the screen, after that, the entire painting on the virtual memory device was transferred to the printer by
calling the Printer Manager of Windows. During the printing procedure processing, an "Abort Printing" dialog box was always presented on the screen. This allowed the user to cancel the printing job at any time. If a user pushed the "OK" button in the "Abort Printing" dialog box, or an error occurred while the contents were transferred to the printer, a message box appeared on the screen indicating a printing error code, and the printing job was terminated.
CHAPTER V
TEST AND VALIDATION

After completing the entire software package development, a simple experiment was conducted to access the efficiency and practicability of this package.

5.1 A SIMPLE EXPERIMENT CONDUCTED AS A PRACTICABILITY TEST

The package had been tested in a class with six graduate students who took ISE 549 in summer session 1993 as subjects. The objective of the experiment was to investigate the subjects' performance in a preview tracking task under variable conditions of preview time, time lag, and input control gain by using the preview tracking program, SIMULATOR.

5.1.1 Experimental design and procedure

The experimental design chosen for this task was a full factorial design with six replications (subjects), two preview times, two time lags, and two input gains. Thus, there were totals of \(2 \times 2 \times 2 = 8\) experimental conditions. The chosen preview times were 0.2 seconds and 2.0 seconds. The time lags were 0 seconds and 1.0 second and the two extreme input gains were 1 and 7. In order to eliminate the learning
curve effect, eight experimental conditions were assigned to
eight runs randomly by the experimenter. The order of the
experimental runs for each subject was listed in table 5.1.
Where S1 through S6 were the subjects one through six, and
R1 through R8 were the run numbers that the subjects ran in
order. The number C1..C8 were the eight experimental
conditions which were listed below.

Table 5.1. The list of experimental runs

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>C6</td>
<td>C2</td>
<td>C7</td>
<td>C4</td>
<td>C3</td>
<td>C8</td>
<td>C1</td>
<td>C5</td>
</tr>
<tr>
<td>S2</td>
<td>C3</td>
<td>C5</td>
<td>C7</td>
<td>C8</td>
<td>C6</td>
<td>C4</td>
<td>C2</td>
<td>C1</td>
</tr>
<tr>
<td>S3</td>
<td>C5</td>
<td>C3</td>
<td>C6</td>
<td>C1</td>
<td>C7</td>
<td>C2</td>
<td>C8</td>
<td>C4</td>
</tr>
<tr>
<td>S4</td>
<td>C8</td>
<td>C1</td>
<td>C3</td>
<td>C2</td>
<td>C4</td>
<td>C5</td>
<td>C6</td>
<td>C7</td>
</tr>
<tr>
<td>S5</td>
<td>C2</td>
<td>C4</td>
<td>C6</td>
<td>C1</td>
<td>C8</td>
<td>C7</td>
<td>C5</td>
<td>C3</td>
</tr>
<tr>
<td>S6</td>
<td>C1</td>
<td>C6</td>
<td>C3</td>
<td>C7</td>
<td>C5</td>
<td>C4</td>
<td>C2</td>
<td>C8</td>
</tr>
</tbody>
</table>

C1 = Gain: 1, Lag: 0 second, Preview Time: 0.2 second.
C2 = Gain: 1, Lag: 0 second, Preview Time: 2.0 second.
C3 = Gain: 1, Lag: 1.0 second, Preview Time: 0.2 second.
C4 = Gain: 1, Lag: 1.0 second, Preview Time: 2.0 second.
C5 = Gain: 7, Lag: 0 second, Preview Time: 0.2 second.
C6 = Gain: 7, Lag: 0 second, Preview Time: 2.0 second.
C7 = Gain: 7, Lag: 1.0 second, Preview Time: 0.2 second.
C8 = Gain: 7, Lag: 1.0 second, Preview Time: 2.0 second.

For all above experimental conditions, the relative
curve rate was set at 60% and the object speed was set at 30
m.p.h.. The road(two lanes) width used in the experiment
was set at 30 feet. The object shape was a circle. Each
experimental run took 60 seconds. The subject was given the chance to adapt to the new experimental condition by allowing a practicing time of 10 seconds (therefore, data of the first 10 seconds in every experimental run was ignored).

5.1.2 Experimental results

The experimental results were illustrated in table 5.2 and figure 5.1. The values used in table 5.2 and figure 5.1 were the percentage of incorrect control (off road) of the experimental time. From table 5.2, it might be seen that the tracking performance was better when the gain was set to 1 than when the gain was set to 7. When the time lag was set to 1.0 second, it appeared to be very difficult for the subject to prevent the object from moving off the road. There was no statistically significant difference in tracking performance for a preview time of 0.2 seconds and 2.0 seconds. (see Figure 5.2)

Table 5.2 Experimental results of 6 subjects and 8 experimental conditions

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>7.60</td>
<td>8.80</td>
<td>57.80</td>
<td>49.20</td>
<td>5.60</td>
<td>6.80</td>
<td>84.80</td>
<td>81.80</td>
</tr>
<tr>
<td>S2</td>
<td>2.60</td>
<td>10.20</td>
<td>58.20</td>
<td>70.00</td>
<td>32.40</td>
<td>6.00</td>
<td>84.60</td>
<td>82.00</td>
</tr>
<tr>
<td>S3</td>
<td>5.50</td>
<td>3.00</td>
<td>58.60</td>
<td>31.60</td>
<td>12.40</td>
<td>26.00</td>
<td>84.60</td>
<td>81.80</td>
</tr>
<tr>
<td>S4</td>
<td>2.00</td>
<td>0.80</td>
<td>53.80</td>
<td>58.80</td>
<td>7.20</td>
<td>12.20</td>
<td>87.00</td>
<td>80.20</td>
</tr>
<tr>
<td>S5</td>
<td>13.00</td>
<td>6.00</td>
<td>69.20</td>
<td>64.40</td>
<td>28.00</td>
<td>34.40</td>
<td>79.40</td>
<td>87.20</td>
</tr>
<tr>
<td>S6</td>
<td>1.40</td>
<td>1.20</td>
<td>61.80</td>
<td>71.20</td>
<td>25.80</td>
<td>24.60</td>
<td>90.60</td>
<td>89.40</td>
</tr>
<tr>
<td>AVG</td>
<td>5.35</td>
<td>5.00</td>
<td>59.90</td>
<td>57.53</td>
<td>18.57</td>
<td>18.33</td>
<td>85.17</td>
<td>83.73</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>4.43</td>
<td>3.96</td>
<td>5.22</td>
<td>15.05</td>
<td>11.56</td>
<td>11.65</td>
<td>3.66</td>
<td>3.66</td>
</tr>
</tbody>
</table>


• = the mean value
Upper ◊ = the mean value + the standard deviation
Lower ◊ = the mean value - the standard deviation

Figure 5.1 Experimental results illustrated in mean values and standard deviations
A- ==> INPUT GAIN 1
A+ ==> INPUT GAIN 7
B- ==> TIME LAG 0 sec.
B+ ==> TIME LAG 1 sec.
C- ==> PREVIEW TIME 0.2 sec.
C+ ==> PREVIEW TIME 2.0 sec.

Y-AXIS: OFF ROAD RATE
X-AXIS: EXPERIMENTAL VARIABLE

Note: The figure was dumped from the program STATS, developed by IBM Inc.

Figure 5.2 The main effect of three variables
Table 5.3 ANOVA table of the experimental result analysis

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>Sum of Square</th>
<th>D.F.</th>
<th>Mean Square</th>
<th>F-value</th>
<th>Signif. of F</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAIN(G)</td>
<td>4600.125</td>
<td>1</td>
<td>4600.125</td>
<td>74.751</td>
<td>0.00...</td>
</tr>
<tr>
<td>LAG(L)</td>
<td>42763.107</td>
<td>1</td>
<td>42763.107</td>
<td>694.894</td>
<td>0.00...</td>
</tr>
<tr>
<td>PREVIEW</td>
<td>12.505</td>
<td>1</td>
<td>12.505</td>
<td>0.203</td>
<td>&gt; 0.25</td>
</tr>
<tr>
<td>TIME(P)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(G)x(L)</td>
<td>454.485</td>
<td>1</td>
<td>454.485</td>
<td>7.385</td>
<td>0.011</td>
</tr>
<tr>
<td>(G)x(P)</td>
<td>1.367</td>
<td>1</td>
<td>1.367</td>
<td>0.022</td>
<td>&gt; 0.25</td>
</tr>
<tr>
<td>(L)x(P)</td>
<td>9.275</td>
<td>1</td>
<td>9.275</td>
<td>0.151</td>
<td>&gt; 0.25</td>
</tr>
<tr>
<td>(G)x(L)x(P)</td>
<td>0.200</td>
<td>1</td>
<td>0.200</td>
<td>0.003</td>
<td>&gt; 0.25</td>
</tr>
<tr>
<td>REPLICATION*</td>
<td>798.171</td>
<td>5</td>
<td>159.634</td>
<td>2.594</td>
<td>0.045</td>
</tr>
<tr>
<td>Residual</td>
<td>2153.850</td>
<td>35</td>
<td>61.539</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>50793.085</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: (*) 6 Replications => 6 Subjects

From the ANOVA table (see Table 5.3), it was easy to find that time lag was the most distinguishable from the other factors and interactions. Its main effect was too strong that might make the other factors' one way effects and interactions hard to be observed. In this ANOVA table, the other factors, input gain and subject, and the interaction between gain and lag were also significant in 95% confidence interval, but not as significant as time lag.

All the experimental results listed in table 5.2 were obtained under low object speed (only 30 m.p.h.) and a fairly
wide road. If the experimental variables were changed, the experimental results could be totally different.

After interviewing six subjects, it became evident that this preview tracking experiment was not easy. The subjects indicated that it was easy for them to control the object under normal conditions (when the input gain is one and the time lag is zero seconds). However, when the input gain was set to seven, these subjects reported more difficulties. One reason for the difficulties may be explained by the very sensitive MAXX II Yoke. When the time lag was set to one second, the subjects also reported an uneasiness in controlling the object. This might be resulted from the fact that such a condition was unusual in their daily lives. However, the subjects agreed that the package was suitable for experimental and educational purposes.

On the experimenters side, the experimenter pointed out that the package was easy to use, but it appeared slightly too many experimental variables needed to be set up at the beginning of the experiment. However, in order to reach the system requirements and keep the flexibility of the package, it was inevitable for the package to require that many variables.
5.2 VALIDATION OF RELATIVE CURVE FREQUENCY

The relative curve frequency determined the percentage of curves produced during the experiment. The package checked whether the curve-selection random number generated was within a given bound. If the value was greater than a certain value, then a straight section was generated. If the random number was lower than the certain value, then the curve section was generated.

In the program developed by Viswanathan and Zwahlen (1986), the random number determined not only the shape of the road section, but also its length, which resulted in an obvious difference between the curve rate generated by the computer and the curve rate requested by the experimenter. In this new package, the length of every road section is fixed to be fifty feet (200 pixels on the screen), which eliminated the above problem. There were 10 initial seeds picked up randomly to validate the relative curve rate generated by SIMULATOR. After validating the relative curve rate, it was verified that there was no considerable difference between the actual curve rate and the curve rate requested. The validation results were shown in Figure 5.3.
Figure 5.3 Validation of relative curve rate
5.3 VALIDATION OF MEAN VALUE AND STANDARD DEVIATION

In the experimental results report (.RST file), the mean and the standard deviation of the lateral position were listed. The former was the object center position with respect to the right road edge (a negative value means the object is inside the road), and the latter was the standard deviation of the distance between the object center position and the right road edge.

The object's center position depended on the input value from the MAXX II Yoke. Since it was very hard for users to stably hold the yoke, a new function was added to the function YokeInput to verify and validate the experimental results report generated by the package. The new function was used to generate expected constants which were used to replace the input values obtained from the yoke. Thus, the experimenter could obtain the expected car route to generate expectable output mean and standard deviation. Then, the mean and standard deviation values generated by this package would be compared with the results got by hand calculations. Table 5.4 illustrated the verification results.

After implementing the new function to test the output values, the results showed that there was no significant difference in the standard deviation obtained by the values
generated through the package and the values obtained by manual calculations. The difference of the mean value between package-generated and manual-calculated was 0.001 feet, and the difference of the standard deviation was 0.002 feet.

Table 5.4 The comparison of the mean and standard deviation calculated by hands and by the analysis package

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Calculation by hands</th>
<th>Generated by comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std. dev.</td>
</tr>
<tr>
<td>1</td>
<td>-11.01082</td>
<td>3.7521038</td>
</tr>
<tr>
<td>2</td>
<td>-11.92045</td>
<td>3.3432156</td>
</tr>
<tr>
<td>3</td>
<td>-12.84729</td>
<td>3.964298</td>
</tr>
<tr>
<td>4</td>
<td>-10.95236</td>
<td>3.422017</td>
</tr>
<tr>
<td>5</td>
<td>-12.09467</td>
<td>3.7654158</td>
</tr>
</tbody>
</table>

Note: - meant the object position was left to the right edge of the road
CHAPTER VI

CONCLUSIONS

The software package SIMULATOR/TRACER which could be used to measure the human response to a preview tracking task had been successfully developed and tested. The programs were designed to satisfy all system requirements that were described in an earlier chapter.

The designed package had been successfully tested on an Auro 486 DX/33 ISA system which was equipped with 8MB RAM, a Conner 340 MB IDE hard disk (70 MB free space), a Diamond SpeedStar 24X video accelerator card, a Creative Lab's Sound Blaster card, a Kraft dual-port game card, and a 14" SVGA monitor. The package fulfilled all target requirements set for the thesis project with the exception of the animation refresh frequency which was a little slower than the targeted 14 fps. The slower animation frequency of 10 fps was the result of the tradeoff between system performance and a well-designed graphical user interface written for MS Windows. User friendliness was one of the most important requirements in this thesis. This package achieved the objective of this thesis project in an efficient and effective manner under the constraints of time and resources.
6.1 SUGGESTIONS FOR FUTURE RESEARCH

Approaching to the end of this project, some areas of special interest are highlighted.

1. The technique of Multimedia Personal Computer (MPC) has been developed maturely and was now available for a reasonable investment. By using these enhanced video images, stereo audio sounds, and photographic processing techniques, the subjects could be exposed to more realistic driving conditions.

2. A three-dimensional or perspective model display could be developed. Using the current top-view model with the restricted height of the client area, it is impossible to realistically simulate the object driving at high speeds with long preview times. The perspective model display could help to break through this restriction.

3. A larger screen with better resolution (such as 800 pixels x 600 pixels or higher) would be highly recommended for the experiment. Since the maximum preview time and preview distance were restricted by the resolution and size of the screen (normally, 640 pixels x 480 pixels in 14" VGA monitor) in the top-view system, therefore, a larger screen could make the system more flexible and less restrictive.
4. Computer network systems with well designed graphical user interfaces had been introduced recently (such as Windows for Workgroup or Windows NT). By adopting such a computer network system, it would be possible to execute the preview tracking experiment on a main computer, while the client computers could be used to store experimental data, to display the controlled variables, and to analyze or plot the experimental conditions in real time. This would greatly increase the convenience of the system operation and the system's overall performance.
BIBLIOGRAPHY


Appendix A

Source Code File (SIMU.C) of SIMULATOR

with Detailed Description
The Source Code of Simulator

#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include "simu.h"

// Prototype of functions
int PASCAL WinMain ( HANDLE, HANDLE, LPSTR, int );
// Main function
long FAR PASCAL MainWndProc ( HWND, UINT, UINT, LONG );
// Main Window procedure
long FAR PASCAL SimmWndProc ( HWND, UINT, UINT, LONG );
// Child Window procedure
long FAR PASCAL CRWndProc ( HWND, UINT, UINT, LONG );
// Child Window procedure
BOOL FAR PASCAL AboutProc ( HWND, UINT, UINT, LONG );
// Dialog Box procedure
BOOL FAR PASCAL Dial1Proc ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL Dial1Proc ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL Dial2Proc ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL Dial3Proc ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL Dial4Proc ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL FPDPProc ( HWND, UINT, UINT, LONG );
BOOL FAR PASCAL FPTProc ( HWND, UINT, UINT, LONG );

void InitialRandomNumber(void);
void DrawRoad(void);
void CollectData(void);
void YokeInput(void);
void CarPosition(void);
void SetXValue(void);
void SelectShape(void);

int nLRAN[215];
int nRANI, nNOCorS;

HANDLE hInst; // Instance of The Windows Application
HWND hMainWnd, hSimmWnd, hChildWnd, hCRWnd;
// Handles of Child Window
FARPROC lpAboutProc, lpDial1Proc, lpDial1Proc, lpDial2Proc;
FARPROC lpDial3Proc, lpDial4Proc;
// Long Pointers to Dialog Box Procedure
int sWidth, sHeight, cWidth, cHeight;
// Window's Width & Height
char szMainCaption[] = "Simulator 1.0";
// Main Window Caption
char szSubCaption[26] = " ";
// Subject ID as Sub Window Caption
char szTextOut[50], cBuffer[50];
// Text Output Buffer
HDC hDC, hMemDC;  // Device & Memory Device Control ID
PAINTSTRUCT ps;   // Paint Structure
RECT rRect;  // Rectangle Structure
HMENU hMenu = 0;  // Handle of Menu (Menu ID)
HPEN hPen, hOldPen;  // Handle of Pens
HBRUSH hBrush, hOldBrush;  // Handle of Brushes
HBITMAP hBitmap1, hOldBitmap;  // Handle of Bitmaps

int nRoadWidthft = 24;  // Road Width in ft.
int nRoadWidthPixel;  // Road Width in Pixel
// Scale: 1ft = 4pixels

// Variables set up in Dilog Boxes
int nPreDist = 80;  // Preview Distance in ft
int nSimultSpeed = 60;  // Object Speed
int nSpeedTemp = 4;  // Temp. Object Speed
int nSpeedBuff;
int nRoadWidthfttemp = 50;
int nPreDisttemp = 80;
int nSpeedVariablefttemp = 50;  // Variable of Object Speed
int nSpeedVariableTmp = 0;

BOOL bBool1, bBool2, bBool3, bDial10, bDial1, bDial2, bDial3,
bData1, bDial4;
// Dialog Procedure Return Value

int nIteration;  // Experiment Time
int nIndex;  // Index Buffer
int nXPos, nYPos;  // Road Position in X and Y Coordinates
static int nXDisplayPos[410], nCLorno[410];
// Data Array to Put Road Edge and Center Line
// X-Coordinate positions

int nSeed;  // Initila Seed to Generate Random Number
int nSeeddtmp = 0;
int nSeedInit = 0;
int nRanIndexLast = 0;  // Generated Random Number
int nRanIndexNow, nRanIndexNext;
float fLineorCurve;  // Relative Curve rate
int nOffRoadTime=0; // Total off Road Time and Distance
long nOffRoadDist=0;
float fOffRoadDist;
int nScrollPixels; // Pixels to Scroll Down
int nBrake = 0; // Brake is on or off, 1 is on
int hFileHandle; // To Indicate whether the file is found or not
int nLeftRoad[3001], nCarPos[3001], nCarSpeed[3001],
    nXCarPosTemp; // Data Array to save data
char szDataBuff[50]; // OutPut Buffer
char szFileName[80], szFileExtention[90]; // File Name
char szDirectory[60], szDirectoryTmp[60]; // Directory Name
char szFilelnfo[90], szFileRst[90], szFileRan[90];
char szPathTmp[60];

// Variables in Dial3
int nShapeTmp, nObjectShape = 907;
int nCurves=0;
float fCurves;
int nLags=0;
int nGains=1;
int nDrawCL=910; // Draw Center Line Or Not
int nCLTmp=0;
int nCLINDEX=80;

//Object Width & Length
int nObjectLength, nObjectWidth;

//Yoke Input
int nYokeMax=71;
int nYokeMin=9;
int nYokeCen=40;
int nYokeInput=40; // Default Yoke Input
int nYokeBuffer, nYokeTmpBuffer;
int nLastYokeInput, nLagRunning = 0;

//Pedal Input
int nPedalInput;
int nIdeaSpeed=0;

//Accumulated Car Position
int nCP[7];

//Variables of Statistical Analysis
float fRCMeanTmp, fRCMean;
// Mean Value of Object to Center Line
float fRBMean, fRBMeanTmp;
// Mean Value ............ to Right Road Edge
float fAvgSpeed, fAvgSpeedTmp; // Average Object Speed
float fRCSTDTmp, fRCSTD, fRBSTD, fRBSTDTmp;
// Standard Deviation of ....
long int nRCSTD0000, nRBSTD0000, nRCMean00, nRBMean00,
   nAvgSpeed000;
int nRBID, nRCID; // If nRBID = 0, fRBMean is positive

//Variables for Road Generation.
int nCL[7][200], nRadius;
int n100, n10, n1, nSecIndex;
char sz100[1], sz10[1], sz1[1];
int nLastShape, nNowShape, nFinalXPos;
int nLR; // Right Side or Left Side
int nRX, nCX, nLX;

//Variables in Dialog Box "Dial4"
int nETime = 801; // Experiments Time
int nITime = 851; // Subject Adapt time
int nETimeTmp = 1;
int nITimeTmp = 1;
int nCurveDir = 870;
int nCurveDirTmp = 0;
int nIgnoreTimes = 50;
int nRunTimes;

//Variables in Dialog Box "Dial1"
int nPreviewSelect=1100; // Preview Time or Distance
int nPreviewSelectTmp=0;
int nPreviewTime=1402;
int nPreviewTimeTmp=0;
int nSeeDist;

//Variables in Dialog Box "FPT" and "FPD"
int nLSpeed[8]={0,0,0,0,0,0,0,0};
int nHSpeed[8]={120,120,75,60,30,18,12,9};
int nSemSpeedVariable=750;
int nSemPreviewTime=1402;
int nSemSimultSpeed=60;
int nSemPreDist=80;

//for Time Lag
int nLag[12], nCoun, nLagCoun;

int PASCAL WinMain ( hInstance, lpPrevInstance, lpCmdLine,
                     nCmdShow )
HANDLE hInstance;
HANDLE lpPrevInstance;
LPSTR lpCmdLine;
int nCmdShow;
{
  MSG      msg;
  HWND     hWnd;
  WNDCLASS wc;
  sWidth = GetSystemMetrics( SM_CXSCREEN ) ;
  sHeight = GetSystemMetrics( SM_CYSCREEN ) ;
  cWidth = GetSystemMetrics( SM_CXFULLSCREEN ) ;
  cHeight = GetSystemMetrics( SM_CYFULLSCREEN ) ;

  // Register the Windows application

  if ( !hPrevInstance )
    
    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon ( hInstance, "THESISICON" );
    wc.hCursor = LoadCursor ( NULL, IDC_ARROW );
    wc.hbrBackground = GetStockObject ( WHITE_BRUSH );
    wc.lpszMenuName = "SimmMenu";
    wc.lpszClassName = "SimmClass";

    if ( !RegisterClass ( &wc ) )
        return ( FALSE );

    wc.lpfnWndProc = SimmWndProc ;
    wc.hInstance = hInstance ;
    wc.hCursor = LoadCursor ( NULL, IDC_ARROW );
    wc.lpszMenuName = NULL ;
    wc.hIcon = LoadIcon( hInstance, "THESISICON" );
    wc.lpszClassName = "NewClass" ;
    wc.hbrBackground = GetStockObject ( WHITE_BRUSH ) ;
    wc.cbClsExtra = 0 ;
    wc.cbWndExtra = 0 ;

    if ( !RegisterClass ( &wc ) )
        return ( FALSE );

    wc.lpfnWndProc = CRWndProc ;
    wc.hInstance = hInstance ;
    wc.hCursor = LoadCursor ( NULL, IDC_ARROW );
    wc.lpszMenuName = NULL ;
    wc.hIcon = LoadIcon( hInstance, "THESISICON" );
    wc.lpszClassName = "CRClass" ;
    wc.hbrBackground = GetStockObject ( WHITE_BRUSH ) ;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;

if (!RegisterClass (&wc))
    return ( FALSE );

hInst = hInstance;

// Call MainWndProc() to create main window

hMainWnd = CreateWindow( "SimmClass",
        szMainCaption,
        WS_OVERLAPPEDWINDOW,
        0,
        0,
        sWidth,
        sHeight,
        NULL,
        NULL,
        hInstance,
        NULL
    );

if ( !hMainWnd )
    return ( FALSE );

ShowWindow ( hMainWnd, nCmdShow );
UpdateWindow( hMainWnd );

// create the initial child window

hCRWnd = CreateWindow ( "CRClass",
        "About SIMULATOR 1.0",
        WS_CHILD | WS_Caption | WS_SYSMENU
        | WS_MAXIMIZEBOX |
        WS_MINIMIZEBOX,
        0, 0, cWidth, cHeight, hMainWnd,
        NULL, hInst, NULL
    );

// Show the child window

if ( !hCRWnd )
    return ( FALSE );

ShowWindow ( hCRWnd, SW_SHOWNORMAL );
UpdateWindow ( hCRWnd );
// Enter message loop

while ( GetMessage( &msg, NULL, NULL, NULL ) )
{
    TranslateMessage( &msg );
    DispatchMessage( &msg );
}
return ( msg.wParam );

MainWndProc()

long FAR PASCAL MainWndProc ( hWnd, message, wParam, lParam )
HWND hWnd;
UINT message;
UINT wParam;
LONG lParam;
{
    switch ( message )
    {
        case WM_CREATE:

// Look for data (.INI) files and read data
// every 3 digits represent a number
// There are 7 .INI files.
// The data will be put in nCL[]

    hFileHandle=_lopen((LPSTR)"SETUP.INI",0);
    if ( hFileHandle != -1 )
    {
        _llseek( hFileHandle, 0, 0 );
        _lread( hFileHandle, (LPSTR)sz100, 1 );
        _llseek( hFileHandle, 1, 0 );
        _lread( hFileHandle, (LPSTR)sz10, 1 );
        _llseek( hFileHandle, 2, 0 );
        _lread( hFileHandle, (LPSTR)sz1, 1 );
        n100=(int)(*sz100)-48;
        n10=(int)(*sz10)-48;
        n1=(int)(*sz1)-48;
        nRadius = n100*100+n10*10+n1;
    }

    hFileHandle=_lopen((LPSTR)"LB.INI",0);
    if ( hFileHandle != -1 )
    {
        for (nIndex=0; nIndex<40; nIndex++)
```c
{ 
    _llseek( hFileHandle, 17*nIndex+0, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+1, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+2, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)(*sz100)-48;
    n10=(int)(*sz10)-48;
    n1=(int)(*sz1)-48;
    nCL[1][nIndex*5] = n100*100+n10*10+n1;

    _llseek( hFileHandle, 17*nIndex+3, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+4, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+5, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)(*sz100)-48;
    n10=(int)(*sz10)-48;
    n1=(int)(*sz1)-48;
    nCL[1][nIndex*5+1] = n100*100+n10*10+n1;

    _llseek( hFileHandle, 17*nIndex+6, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+7, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+8, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)(*sz100)-48;
    n10=(int)(*sz10)-48;
    n1=(int)(*sz1)-48;
    nCL[1][nIndex*5+2] = n100*100+n10*10+n1;

    _llseek( hFileHandle, 17*nIndex+9, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+10, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+11, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)(*sz100)-48;
    n10=(int)(*sz10)-48;
    n1=(int)(*sz1)-48;
    nCL[1][nIndex*5+3] = n100*100+n10*10+n1;

    _llseek( hFileHandle, 17*nIndex+12, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+13, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
};
```
_llseek( hFileHandle, 17*nIndex+14, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)((*sz100)-48);
n10=(int)((*sz10)-48);
n1=(int)((*sz1)-48);
nCL[1][nIndex*5+4] = n100*100+n10*10+n1;
}
}

hFileHandle=_lopen((LPSTR)"RB.INI",0);
if ( hFileHandle != -1 )
{
  for (nIndex=0; nIndex<40; nIndex++)
  {
    _llseek( hFileHandle, 17*nIndex+0, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+1, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+2, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)((*sz100)-48);
    n10=(int)((*sz10)-48);
    n1=(int)((*sz1)-48);
    nCL[2][nIndex*5] = n100*100+n10*10+n1-
nRadius;

    _llseek( hFileHandle, 17*nIndex+3, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+4, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+5, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)((*sz100)-48);
    n10=(int)((*sz10)-48);
    n1=(int)((*sz1)-48);
    nCL[2][nIndex*5+1] = n100*100+n10*10+n1-
nRadius;

    _llseek( hFileHandle, 17*nIndex+6, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+7, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+8, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)((*sz100)-48);
    n10=(int)((*sz10)-48);
    n1=(int)((*sz1)-48);
    nCL[2][nIndex*5+2] = n100*100+n10*10+n1-
nRadius;
_llseek( hFileHandle, 17*nIndex+9, 0 );
_lread( hFileHandle, (LPSTR)sz100, 1 );
_ llseek( hFileHandle, 17*nIndex+10, 0 );
_lread( hFileHandle, (LPSTR)sz10, 1 );
_ llseek( hFileHandle, 17*nIndex+11, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int) (*sz100)-48;
n10=(int) (*sz10)-48;
n1=(int) (*sz1)-48;
nCL[2][nIndex*5+3] = n100*100+n10*10+n1-
nRadius;

_ llseek( hFileHandle, 17*nIndex+12, 0 );
_lread( hFileHandle, (LPSTR)sz100, 1 );
_ llseek( hFileHandle, 17*nIndex+13, 0 );
_lread( hFileHandle, (LPSTR)sz10, 1 );
_ llseek( hFileHandle, 17*nIndex+14, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int) (*sz100)-48;
n10=(int) (*sz10)-48;
n1=(int) (*sz1)-48;
nCL[2][nIndex*5+4] = n100*100+n10*10+n1-
nRadius;
}

hFileHandle=_lopen((LPSTR)"LU.INI",0);
if ( hFileHandle != -1 ){
    for ( nIndex=0; nIndex<40; nIndex++ )
    {
        _llseek( hFileHandle, 17*nIndex+0, 0 );
        _lread( hFileHandle, (LPSTR)sz100, 1 );
        _ llseek( hFileHandle, 17*nIndex+1, 0 );
        _lread( hFileHandle, (LPSTR)sz10, 1 );
        _ llseek( hFileHandle, 17*nIndex+2, 0 );
        _lread( hFileHandle, (LPSTR)sz1, 1 );
        n100=(int) (*sz100)-48;
        n10=(int) (*sz10)-48;
        n1=(int) (*sz1)-48;
        nCL[3][nIndex*5] = n100*100+n10*10+n1;
    }
}
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[3][nIndex*5+1] = n100*100+n10*10+n1;

_llseek( hFileHandle, 17*nIndex+6, 0 );
_lread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+7, 0 );
_lread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+8, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[3][nIndex*5+2] = n100*100+n10*10+n1;

_llseek( hFileHandle, 17*nIndex+9, 0 );
_lread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+10, 0 );
_lread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+11, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[3][nIndex*5+3] = n100*100+n10*10+n1;

_llseek( hFileHandle, 17*nIndex+12, 0 );
_lread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+13, 0 );
_lread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+14, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[3][nIndex*5+4] = n100*100+n10*10+n1;

hFileHandle=_lopen((LPSTR)"LD.INI",0);
if ( hFileHandle != -1 )
{
    for (nIndex=0; nIndex<40; nIndex++)
    {
        _llseek( hFileHandle, 17*nIndex+0, 0 );
        _lread( hFileHandle, (LPSTR)sz100, 1 );
        _llseek( hFileHandle, 17*nIndex+1, 0 );
        _lread( hFileHandle, (LPSTR)sz10, 1 );
        _llseek( hFileHandle, 17*nIndex+2, 0 );
    }
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[4][nIndex*5] = n100*100+n10*10+n1-
nRadius;

_llseek( hFileHandle, 17*nIndex+3, 0 );
_llread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+4, 0 );
_llread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+5, 0 );
_llread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[4][nIndex*5+1] = n100*100+n10*10+n1-
nRadius;

_llseek( hFileHandle, 17*nIndex+6, 0 );
_llread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+7, 0 );
_llread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+8, 0 );
_llread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[4][nIndex*5+2] = n100*100+n10*10+n1-
nRadius;

_llseek( hFileHandle, 17*nIndex+9, 0 );
_llread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+10, 0 );
_llread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+11, 0 );
_llread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[4][nIndex*5+3] = n100*100+n10*10+n1-
nRadius;

_llseek( hFileHandle, 17*nIndex+12, 0 );
_llread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+13, 0 );
_llread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+14, 0 );
_llread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[4][nIndex*5+4] = n100*100+n10*10+n1-
nRadius;
}
}

hFileHandle=_lopen((LPSTR)"RU.INI",0);
if ( hFileHandle != -1 )
{
for (nIndex=0; nIndex<40; nIndex++)
{
  _llseek( hFileHandle, 17*nIndex+0, 0 );
  _lread( hFileHandle, (LPSTR)sz100, 1 );
  _llseek( hFileHandle, 17*nIndex+1, 0 );
  _lread( hFileHandle, (LPSTR)sz10, 1 );
  _llseek( hFileHandle, 17*nIndex+2, 0 );
  _lread( hFileHandle, (LPSTR)sz1, 1 );
  n100=(int)(*sz100)-48;
  n10=(int)(*sz10)-48;
  n1=(int)(*sz1)-48;
  nCL[5][nIndex*5] = n100*100+n10*10+n1-
nRadius;
  _llseek( hFileHandle, 17*nIndex+3, 0 );
  _lread( hFileHandle, (LPSTR)sz100, 1 );
  _llseek( hFileHandle, 17*nIndex+4, 0 );
  _lread( hFileHandle, (LPSTR)sz10, 1 );
  _llseek( hFileHandle, 17*nIndex+5, 0 );
  _lread( hFileHandle, (LPSTR)sz1, 1 );
  n100=(int)(*sz100)-48;
  n10=(int)(*sz10)-48;
  n1=(int)(*sz1)-48;
  nCL[5][nIndex*5+1] = n100*100+n10*10+n1-
nRadius;
  _llseek( hFileHandle, 17*nIndex+6, 0 );
  _lread( hFileHandle, (LPSTR)sz100, 1 );
  _llseek( hFileHandle, 17*nIndex+7, 0 );
  _lread( hFileHandle, (LPSTR)sz10, 1 );
  _llseek( hFileHandle, 17*nIndex+8, 0 );
  _lread( hFileHandle, (LPSTR)sz1, 1 );
  n100=(int)(*sz100)-48;
  n10=(int)(*sz10)-48;
  n1=(int)(*sz1)-48;
  nCL[5][nIndex*5+2] = n100*100+n10*10+n1-
nRadius;
}
hFileHandle=_lopen((LPSTR)"RD.INI",0);
if ( hFileHandle != -1 )
{
for (nIndex=0; nIndex<40; nIndex++)
{
    _llseek( hFileHandle, 17*nIndex+0, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+1, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+2, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)(*sz100)-48;
    n10=(int)(*sz10)-48;
    n1=(int)(*sz1)-48;
    nCL[6][nIndex*5+3] = n100*100+n10*10+n1-nRadius;

    _llseek( hFileHandle, 17*nIndex+12, 0 );
    _lread( hFileHandle, (LPSTR)sz100, 1 );
    _llseek( hFileHandle, 17*nIndex+13, 0 );
    _lread( hFileHandle, (LPSTR)sz10, 1 );
    _llseek( hFileHandle, 17*nIndex+14, 0 );
    _lread( hFileHandle, (LPSTR)sz1, 1 );
    n100=(int)(*sz100)-48;
    n10=(int)(*sz10)-48;
    n1=(int)(*sz1)-48;
    nCL[6][nIndex*5+4] = n100*100+n10*10+n1-nRadius;
}
}
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[6][nIndex*5+1] = n100*100+n10*10+n1;

_llseek( hFileHandle, 17*nIndex+6, 0 );
_lread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+7, 0 );
_lread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+8, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[6][nIndex*5+2] = n100*100+n10*10+n1;

_llseek( hFileHandle, 17*nIndex+9, 0 );
_lread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+10, 0 );
_lread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+11, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[6][nIndex*5+3] = n100*100+n10*10+n1;

_llseek( hFileHandle, 17*nIndex+12, 0 );
_lread( hFileHandle, (LPSTR)sz100, 1 );
_llseek( hFileHandle, 17*nIndex+13, 0 );
_lread( hFileHandle, (LPSTR)sz10, 1 );
_llseek( hFileHandle, 17*nIndex+14, 0 );
_lread( hFileHandle, (LPSTR)sz1, 1 );
n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nCL[6][nIndex*5+4] = n100*100+n10*10+n1;
}

for( nIndex=0; nIndex<200; nIndex++ )
{
    nCL[7][nIndex]=0;
}

break;

case WM_DESTROY:
    // Quit the Program
PostQuitMessage(0);
break;

case WM_COMMAND: /* message: from application menu */
switch(wParam)
{
//
// Create a child window
//
// Initial variables' default value

nYokeInput=nYokeCen;
nYokeBuffer=nYokeCen;
nYokeTmpBuffer=nYokeCen;
nLastYokeInput=nYokeCen;
LagRunning = 0;
PedalInput=24;
LagCount=0;
for( nIndex=0; nIndex<12; nIndex++ )
{
  Lag[nIndex]=0;
}

//
// Open Dialog Box 0

lpDia0Proc = MakeProcInstance
  ((FARPROC)Dia0Proc, 
hInst );
bDia0 = DialogBox ( hInst, "Dia0", hWnd, 
  lpDia0Proc );
FreeProcInstance ( lpDia0Proc );

//
// Open Dialog Box 1

lpDia1Proc = MakeProcInstance
  ((FARPROC)Dia1Proc, 
hInst );
bDia1 = DialogBox ( hInst, "Dia1", hWnd, 
  lpDia1Proc );
FreeProcInstance ( lpDia1Proc );

//
// Open Dialog Box 3

if ((BOOL)bDia1 == TRUE)

//
// If Dialog box 1 closed successfully


{lpDia13Proc = MakeProcInstance
 ((FARPROC)Dia13Proc,
hInst);
 bDia13 = DialogBox ( hInst, "Dia13",
hWnd, lpDia13Proc);
FreeProcInstance ( lpDia13Proc );

// Open Dialog Box 4
    if ((BOOL)bDia13 == TRUE)
    {
        lpDia14Proc = MakeProcInstance
 ((FARPROC)Dia14Proc,
hInst);
        bDia14 = DialogBox ( hInst, "Dia14",
hWnd, lpDia14Proc);
        FreeProcInstance ( lpDia14Proc );

// If all dialog boxes return TRUE, open a new child window
    if ((BOOL)bDia14 == TRUE)
    {
        hSimmWnd = CreateWindow ( "NewClass",
            szSubCaption,
            WS_CHILD | WS_CLIPSIBLINGS,
            0,
            0,
            cWidth,
            cHeight,
            hMainWnd,
            NULL,
            hInst,
            NULL
        );

// Show the child window
        if ( !hSimmWnd )
            return ( FALSE );

            ShowWindow ( hSimmWnd, SW_SHOWNORMAL );
        UpdateWindow ( hSimmWnd );
    }
}
break;
Close the main window

case IDM_EXIT :
    DestroyWindow( hWnd );
    break;

Open About Dialog Box

case IDM_ABOUT :
    lpAboutProc = MakeProcInstance( ( FARPROC )
                                   AboutProc, hInst );
    DialogBox( hInst,"Aboutbox", hWnd,
               lpAboutProc );
    FreeProcInstance( lpAboutProc );
    break;
}
break;

default:
    return ( DefWindowProc( hWnd, message, wParam, lParam ) );
}

return ( NULL );

//---------------------------------------------------- - ~ -- --
// CRWndProc()
//----------------------------------------------------------
// CRWndProc()
long FAR PASCAL CRWndProc ( hWnd, message, wParam, lParam )
HWND hWnd;
UINT message;
UINT wParam;
LONG lParam;
{
    HDC hCRDC, hCRMemDC;
    HBITMAP hCRBitmap;

    switch ( message )
    {
    // get the client area size

    case WM_CREATE:
        sWidth = GetSystemMetrics( SM_CXSCREEN );
        sHeight = GetSystemMetrics( SM_CYSCREEN );
        SetTimer( hWnd, 21, 6000, NULL );
        break;
load the bitmap onto the screen

case WM_PAINT:
    hCRDC = GetDC( hWnd );
    hCRBitmap = LoadBitmap( hInst, "CRBITMAP" );
    hCRMemDC = CreateCompatibleDC( hCRDC );
    SelectObject( hCRMemDC, hCRBitmap );
    BitBlt( hCRDC, (sWidth - 500) / 2, (sHeight - 380) / 2,
           500, 380, hCRMemDC, 0, 0, SRCCOPY );
    DeleteDC( hCRMemDC );
    DeleteObject( hCRBitmap );
    ReleaseDC( hWnd, hCRDC );
    ValidateRect( hWnd, NULL );
    break;

// After showing the initial window for 6 sec.,
// close the child window

case WM_TIMER:
    switch ( wParam )
    {
        case 21:
            KillTimer( hWnd, 21);
            DestroyWindow( hWnd );
            break;
        break;
    }
    break;

default:
    return ( DefWindowProc( hWnd, message, wParam, lParam ) );
    return(NULL);
}

//------------------------------------------------------------------------------------------------------------------

SimmWndProc()
//------------------------------------------------------------------------------------------------------------------

long FAR PASCAL SimmWndProc( hWnd, message, wParam, lParam )
    HWND hWnd;
    UINT message;
    UINT wParam;
    LONG lParam;
{
    switch ( message )
    {

case WM_CREATE :

    for( nIndex = 0; nIndex < 200; nIndex++ )
        nLRAN[nIndex]=0;

    nRANI=0;
    nNOCoR$ = 0;

    // When the child window is being created, initialize some
    // variables
    // If you do not restart the program, this program will
    // memorize the values that used in last experiment

    // set time lag
    if( nLags !=0 )
    {
        if( nLags%100 != 0 )
            nLagCoun = nLags/100+1;
        else
            nLagCoun = nLags/100;
    }

    // set experimental time
    switch( nETime )
    {
        case 801:
            nRunTimes=600;
            break;

        case 802:
            nRunTimes=1200;
            break;

        case 803:
            nRunTimes=1800;
            break;

        case 804:
            nRunTimes=2400;
            break;

        case 805:
            nRunTimes=3000;
            break;
    }
// set subject adaptation time

switch( nITime )
{
    case 851:
        nIgnoreTimes=50;
        break;
    case 852:
        nIgnoreTimes=100;
        break;
    case 853:
        nIgnoreTimes=150;
        break;
    case 854:
        nIgnoreTimes=200;
        break;
    case 855:
        nIgnoreTimes=300;
        break;
}

SetRect( &rRect, 11, 11, 510, 420 );

// Create Caption of main window

strcpy((LPSTR)szTextOut, (LPSTR)szMainCaption );
strcat( (LPSTR)szTextOut, (LPSTR)" :: " );
strcat( (LPSTR)szTextOut, (LPSTR)szSubCaption );
SetWindowText( hMainWnd, (LPSTR)szTextOut );

// Disable Menu Item - NEW... when child window is opened

hMenu = GetMenu( hMainWnd );
EnableMenuItem( hMenu, IDM_NEW,
    MF_GRAYED|MF_DISABLED|MF_BYCOMMAND );
EnableMenuItem( hMenu, IDM_ABOUT,
    MF_GRAYED|MF_DISABLED|MF_BYCOMMAND );

// Load Object Bitmap

if( nObjectShape == IDD_SHAPECAR )
{
    hBitmap1 = LoadBitmap( hInst, "CARBITMAP" );
nObjectLength=47;
nObjectWidth=29;
else
{
    hBitmap1 = LoadBitmap( hInst, "CIRCLEBITMAP" );
    nObjectLength=29;
    nObjectWidth=29;
}

    // Save OldBitmap

    hDC = GetDC( hWnd );
    hMemDC = CreateCompatibleDC( hDC );
    hOldBitmap = SelectObject( hMemDC, hBitmap1 );
    ReleaseDC( hWnd, hDC );

    //
    // fCurves = (float)nCurves;
    fLineorCurve = ((float)fCurves/100.0)*256.0 ;
    nRoadWidthPixel=nRoadWidthft*4;
    nXPos=10+(500-nRoadWidthPixel)/2;

    // Initial Position

    // Initial the random number generator

    srand( nSeed );
    nRanIndexNow = random( 256 );
    nRanIndexNext = random( 256 );
    nLRAN[nRANI] = nRanIndexNow;
    if( nRanIndexNext == 0 && fCurves==0 )
        nRanIndexNext=nRanIndexNext+1;
    if( nRanIndexNow == 0 && fCurves==0 )
        nRanIndexNow=nRanIndexNow+1;

    // Scroll speed isScrolling down 7 pixels in every 0.02
    // second when object speed is 60 miles/hour.

    nSpeedBuff = nSimultSpeed;
    nScrollPixels = (7*(float)nSpeedBuff)/12.0;

    nCLornot[409]=1;
    nSecIndex = 200;

    nLR = nCurveDir-870;  // 0------> Right
        // 1------> Left

    nNowShape =0;
    nFinalXPos = nXPos;
    nCX = nXPos;
    nLX = nXPos-nRadius;
    nRX = nXPos+nRadius;
for( nIndex=409; nIndex>=0; nIndex-- )
{
    // Draw Center Line
    nCLINDEX--;  
    if ( nCLINDEX<=-1)  
        nCLINDEX=80;
    if ( nCLINDEX<=50 )  
        nCLOrnot[nIndex]=0;
    else
        nCLOrnot[nIndex]=1;

    // Set X-Coordinate value of the road edge
    SetXValue();
}

if( nPreviewSelect == IDD_FT)  
    nSeeDist=(nPreviewTime-1400)*nScrollPixels;
else
    nSeeDist=nPreDist*4;

nRoadWidthPixel=nRoadWidthft*4;
nXCarPosTemp = nXDisplayPos[nSeeDist]+((nRoadWidthPixel)*3)/4-(nObjectWidth-1)/2;

// because car width is nObjectWidth; so,
// (29-1)/2 = (nObjectWidth-1)/2

// Create a timer

SetTimer( hWnd, 1, 100, NULL );
break;

case WM_TIMER:
    switch ( wParam )
    {
        case 1:

            // Timer 1 is used to set up everything on the screen.

            nIteration = 0;
            KillTimer( hWnd, 1);
            hDC = GetDC ( hWnd );

            // Paint the screen

            hBrush = GetStockObject( NULL_BRUSH );
hPen = GetStockObject( BLACK_PEN );
hOldBrush = SelectObject( hDC, hBrush );
hOldPen = SelectObject( hDC, hPen );
Rectangle( hDC, 10, 10, 511, 421 );
Rectangle( hDC, 560, 10, 621, 311 );

Set speed bar scales

wsprintf( cBuffer, "%3i", 150 );
TextOut( hDC, 530, 10, cBuffer, lstrlen( cBuffer ) );
wsprintf( cBuffer, "%3i", 120 );
TextOut( hDC, 530, 70, cBuffer, lstrlen( cBuffer ) );
wsprintf( cBuffer, "%3i", 90 );
TextOut( hDC, 530, 130, cBuffer, lstrlen( cBuffer ) );
wsprintf( cBuffer, "%3i", 60 );
TextOut( hDC, 530, 190, cBuffer, lstrlen( cBuffer ) );
wsprintf( cBuffer, "%3i", 30 );
TextOut( hDC, 530, 250, cBuffer, lstrlen( cBuffer ) );
wsprintf( cBuffer, "%3i", 0 );
TextOut( hDC, 530, 310, cBuffer, lstrlen( cBuffer ) );
TextOut( hDC, 571, 315, "MPH", lstrlen( "MPH" ) );

DrawRoad();

SelectObject( hDC, hOldBrush );
SelectObject( hDC, hOldPen );
DeleteObject( hPen );
DeleteObject( hBrush );

// Draw Speed Bar to show object speed

hBrush = CreateSolidBrush( RGB(255, 0, 0) );
hPen = GetStockObject( NULL_PEN );
hOldBrush = SelectObject( hDC, hBrush );
hOldPen = SelectObject( hDC, hPen );
Rectangle( hDC, 561, 311-(2*nSimultSpeed), 620, 310 );
SelectObject( hDC, hOldBrush );
SelectObject( hDC, hOldPen );
DeleteObject( hPen );
DeleteObject( hBrush );

// Draw Speed Bar (Other Side)
hBrush = CreateSolidBrush(RGB(0, 0, 0));
hPen = GetStockObject(NULL_PEN);
hOldBrush = SelectObject(hDC, hBrush);
hOldPen = SelectObject(hDC, hPen);
Rectangle(hDC, 561, 11, 620, 311-(2*nSimultSpeed));
SelectObject(hDC, hOldBrush);
SelectObject(hDC, hOldPen);
DeleteObject(hPen);
DeleteObject(hBrush);

// Draw Brake Rectangle

if( nBrake == 1 )
hBrush = CreateSolidBrush(RGB(255, 0, 0));
else
hBrush = CreateSolidBrush(RGB(0, 255, 255));

hPen = GetStockObject(BLACK_PEN);

hOldBrush = SelectObject(hDC, hBrush);

hOldPen = SelectObject(hDC, hPen);

Rectangle(hDC, 531, 336, 621, 426);

SelectObject(hDC, hOldBrush);

SelectObject(hDC, hOldPen);

DeleteObject(hPen);

DeleteObject(hBrush);

// Put Object bitmap on the screen

if( nPreviewSelect == IDD_FT)
nSeeDist = (nPreviewTime-1400)*nScrollPixels;
else
nSeeDist = nPreDist*4;

BitBlt(hDC, nXCarPosTemp, 10+nSeeDist, nObjectWidth, nObjectLength, hMemDC, 0, 0, SRCCOPY);

ReleaseDC(hWnd, hDC);

// collect experiment data

CollectData();
SetTimer( hWnd, 2, 500, NULL );
SetTimer( hWnd, 3, 100, NULL );
break;

case 2:
    // Timer 2 is set to draw speed bar, draw brake,
    // check check object position.
    
    hDC = GetDC( hWnd );
    // Draw speed bar
    
    hBrush=CreateSolidBrush(RGB(255,0,0));
hPen = GetStockObject( NULL_PEN );
hOldBrush = SelectObject( hDC, hBrush );
hOldPen = SelectObject( hDC, hPen );
Rectangle( hDC, 561, 311-(2*nSpeedBuff),
        620, 310 );
SelectObject( hDC, hOldBrush );
SelectObject( hDC, hOldPen );
DeleteObject( hPen );
DeleteObject( hBrush );

    hBrush = GetStockObject( BLACK_BRUSH );
hOldBrush = SelectObject( hDC, hBrush );
Rectangle( hDC, 561, 11,
        620, 311-(2*nSpeedBuff) );
SelectObject( hDC, hOldBrush );
DeleteObject( hBrush );

    // Draw brake rectangle
    
    if( nBrake == 0 )
        hBrush=CreateSolidBrush(RGB(0,255,255));
    else
        hBrush=CreateSolidBrush(RGB(255,0,0));

    hPen = GetStockObject( BLACK_PEN );
hOldBrush = SelectObject( hDC, hBrush );
hOldPen = SelectObject( hDC, hPen );
Rectangle( hDC, 531, 336,
        621, 426 );
SelectObject( hDC, hOldBrush );
SelectObject( hDC, hOldPen );
DeleteObject( hPen );
DeleteObject( hBrush );
ReleaseDC( hWnd, hDC );
nRoadWidthPixel=nRoadWidthFt*4;

// Check object position. If it is out of the road, a beep will be generated.
if(
    nCarPos[nIteration]<nLeftRoad[nIteration]| |
    (nCarPos[nIteration]+nObjectWidth)>
    (nLeftRoad[nIteration]+nRoadWidthPixel))
    MessageBeep(0);
    break;

case 3:
    // Calculate the x-coordinate position of road
    nIteration++;
    // If it is out of experiment time, the child window will be closed.
    if ( nIteration > nRunTimes )
    {
        KillTimer( hWnd, 3 );
        SendMessage( hWnd,WM_CLOSE,NULL,NULL );
    }
    // Receive yoke and pedals input
    YokeInput();
    // Calculate x-coordinate Position of the center line and the road edge
    nScrollPixels = (nSpeedBuff*7)/12.0;
    for( nIndex=409; nIndex>=nScrollPixels;
        nIndex-- )
    {
        nCLornot[nIndex]=nCLornot[nIndex-nScrollPixels];
        nXDisplayPos[nIndex] =
            nXDisplayPos[nIndex-nScrollPixels];
    }
for( nIndex=nScrollPixels-1; nIndex>=0; nIndex-- )
{
    nCLINDEX--;  
    if ( nCLINDEX<=-1)  
        nCLINDEX=80;  
    if ( nCLINDEX<=50)  
        nCLornot[nIndex]=0;  
    else  
        nCLornot[nIndex]=1;

    SetXValue();
}

// Call WM_PAINT message
InvalidateRect( hWnd, &rRect, TRUE );
break;
}
break;
case WM_PAINT :
       
hDC = BeginPaint ( hWnd, &ps );

       // decide the preview distance

       if( nPreviewSelect == IDD_FT)  
            nSeeDist=(nPreviewTime-1400)*nScrollPixels;
       else  
            nSeeDist=nPreDist*4;

       // Display Object Bitmap

       BitBlt( hDC,  
             nXCarPosTemp,  
             10+nSeeDist,  
             nObjectWidth, nObjectLength, hMemDC, 0, 0,  
             SRCCOPY );

       // Draw Road

       hBrush = GetStockObject( NULL_BRUSH );
       hPen = GetStockObject( BLACK_PEN );
       hOldBrush = SelectObject( hDC, hBrush );
       hOldPen = SelectObject( hDC, hPen );
       DrawRoad();
       SelectObject( hDC, hOldBrush );
       SelectObject( hDC, hOldPen );
DeleteObject( hPen );
DeleteObject( hBrush );

EndPaint( hWnd, &ps );

// Collect Data and maintain the screen
CollectData();
ValidateRect( hWnd, &rRect );
break;

case WM_DESTROY :

KillTimer( hWnd, 2 );
strcpy((LPSTR)szTextOut, (LPSTR)szMainCaption );
SetWindowText( hWnd, (LPSTR)szTextOut );
hMenu = GetMenu( hWnd );
EnableMenuItem( hWnd, IDM_NEW,
   MF_ENABLED|MF_BYCOMMAND );
EnableMenuItem( hWnd, IDM_ABOUT,
   MF_ENABLED|MF_BYCOMMAND );

// Delete the memory device that created in WM_CREATE
SelectObject( hMemDC, hOldBitmap );
DeleteDC( hMemDC );
DeleteObject( hBitmap );

// Present a dialog box on the screen to ask users to save data
bDatal = BWCCMessageBox( hWnd,
   "Experiment ended !\nDo you want to create a file to save the data ?",
   "Data Save...", MB_ICONQUESTION | MB_YESNO );
if( bDatal == IDYES )
{
   lpDial2Proc = MakeProcInstance(( FARPROC )DlgProc, hInst );
   bDial2 = DialogBox( hInst, "Dial2", hWnd,
   lpDial2Proc );
   FreeProcInstance( lpDial2Proc );
   if( bDial2 == IDOK )
   {
      lstrcpy((LPSTR)szFileExtention, (LPSTR)szPathTmp );
      if((int)szPathTmp[3] !=0)
         lstrcat((LPSTR)szFileExtention, (LPSTR)"\" );
      lstrcat((LPSTR)szFileExtention,(LPSTR)szFileName
      );
   }
lstrcat((LPSTR)szFileExtention, (LPSTR)".dat");

lstrcpy((LPSTR)szFileInfo, (LPSTR)szPathTmp );
if((int)szPathTmp[3]!=0)
lstrcpy((LPSTR)szFileInfo, (LPSTR)"\\");
lstrcat((LPSTR)szFileInfo, (LPSTR)szFileName);
lstrcat((LPSTR)szFileInfo, (LPSTR)".inf");

lstrcpy((LPSTR)szFileRst, (LPSTR)szPathTmp );
if((int)szPathTmp[3]!=0)
lstrcat ((LPSTR)szFileRst, (LPSTR)"\\");
lstrcat((LPSTR)szFileRst, (LPSTR)szFileName);
lstrcat((LPSTR)szFileRst, (LPSTR)".rst");

lstrcpy((LPSTR)szFileRan, (LPSTR)szPathTmp );
if((int)szPathTmp[3]!=0)
lstrcat ((LPSTR)szFileRan, (LPSTR)"\\");
lstrcat((LPSTR)szFileRan, (LPSTR)szFileName);
lstrcat((LPSTR)szFileRan, (LPSTR)".ran");

// create a .DAT file and save data

hFileHandle = _lcreat((LPSTR)szFileExtention,0);
if ( hFileHandle != -1 )
{
    _llseek( hFileHandle, 0, 0);
    wsprintf( szDataBuff,
            "%3.3i%3.3i%3.3i%3.3i",
            nETime-800, nITime-850,
            nRoadWidthft, nObjectWidth);
    _lwrite( hFileHandle, (LPSTR)szDataBuff,
            lstrlen(szDataBuff));

    _llseek( hFileHandle, 12, 0);
    for( nIndex = 0; nIndex < nRunTimes+1;
        nIndex++)
    {
        wsprintf( szDataBuff, "%3.3i%3.3i%3.3i%3.3i",
                nLeftRoad[nIndex],
                nCarPos[nIndex],
                nCarSpeed[nIndex] );
        _lwrite( hFileHandle, (LPSTR)szDataBuff,
                lstrlen(szDataBuff));
        _lwrite( hFileHandle, "\r", 1 );
        _llseek( hFileHandle, 0, 1);
    }
    _lclose( hFileHandle );
}
create a .INF file and save data

    hFileHandle = _lcreat((LPSTR)szFileInfo, 0);
    if ( hFileHandle != -1 )
    {
        _llseek( hFileHandle, 0, 0);
        wsprintf( szDataBuff,
            "%3.3i%3.3i%3.3i%3.3i%9.9i%3.3i%3.3i%3.3i%3.3i%

            nPreDist, nRoadWidthft,
            nSimultSpeed, nSpeedVariable, nSeed,
            nCurves, nLags, nGains,
            nObjectShape, nDrawCL,
            nETime-800,nITime-850,nCurveDir,
            nPreviewSelect-1100, nPreviewTime-1400);
        _lwrite( hFileHandle, (LPSTR)szDataBuff,
            lstrlen(szDataBuff));
        _lwrite( hFileHandle, "\r", 1 );
        _llseek( hFileHandle, 54, 0);
        wsprintf( szDataBuff,
            "%s",
            (LPSTR)szSubCaption);
        _lwrite( hFileHandle, (LPSTR)szDataBuff,
            lstrlen(szDataBuff));
        _lwrite( hFileHandle, "\r", 1 );
        _llseek( hFileHandle, 0, 1);
        _lcloze( hFileHandle );
    }

    calaulate mean, STD value

    fAvgSpeedTmp = 0;
    for( nIndex = nIgnoreTimes+1; nIndex <
        nRunTimes+1; nIndex++)
    {
        fAvgSpeedTmp = fAvgSpeedTmp + nCarSpeed[nIndex];
    }
    fAvgSpeed = fAvgSpeedTmp/(float)(nRunTimes-
        nIgnoreTimes);
    nAvgSpeed000 = (long)(fAvgSpeed*1000);

    fRCMeanTmp = 0;
    fRBMeanTmp = 0;
for( nIndex = nIgnoreTimes+1; nIndex < nRunTimes+1; nIndex++ )
{
    fRCMeanTmp = fRCMeanTmp +
        (float)( nCarPos[nIndex] +
        (float)(0.5*nObjectWidth)) -
        ( nLeftRoad[nIndex] + nRoadWidthft*3 )
    fRBMeanTmp = fRBMeanTmp +
        (float)( nCarPos[nIndex] +
        (float)(0.5*nObjectWidth)) -
        ( nLeftRoad[nIndex] + nRoadWidthft*4 );
}

fRCMean = fRCMeanTmp/(float)(nRunTimes-nIgnoreTimes);
fRBMean = fRBMeanTmp/(float)(nRunTimes-nIgnoreTimes);

fRCSTDTmp = 0;
fRBSTDTmp = 0;
for( nIndex = nIgnoreTimes+1; nIndex < nRunTimes+1; nIndex++ )
{
    fRCSTDTmp = fRCSTDTmp +
        ((float)( nCarPos[nIndex] +
        (float)(0.5*nObjectWidth)) -
        ( nLeftRoad[nIndex] + nRoadWidthft*3 )
    - fRCMean )*
        ((float)( nCarPos[nIndex] +
        (float)(0.5*nObjectWidth)) -
        ( nLeftRoad[nIndex] + nRoadWidthft*3 )
    - fRCMean );
    fRBSTDTmp = fRBSTDTmp +
        ((float)( nCarPos[nIndex]
        +(float)(0.5*nObjectWidth)) -
        ( nLeftRoad[nIndex] + nRoadWidthft*4 )
    - fRBMean )*
        ((float)( nCarPos[nIndex]
        +(float)(0.5*nObjectWidth)) -
        ( nLeftRoad[nIndex] + nRoadWidthft*4 )
    - fRBMean );
}

fRCSTD = sqrt( fRCSTDTmp/(float)(nRunTimes-nIgnoreTimes-1) );
fRBSTD = sqrt( fRBSTDTmp/(float)(nRunTimes-nIgnoreTimes-1) );

if ( fRCMean < 0 )
{

nRCMean00 = 0 - (long)(fRCMean*100);  
nRCID = 1;  
}
else
{
    nRCMean00 = (long)(fRCMean*100);  
nRCID = 0;  
}

if ( fRBMean < 0 )
{
    nRBMean00 = 0 - (long)(fRBMean*100);  
nRBID = 1;  
}
else
{
    nRBMean00 = (long)(fRBMean*100);  
nRBID = 0;  
}

nRCSTDOOOO = (long) (fRCSTD*10000);
nRBSTDOOOO = (long) (fRBSTD*10000);

// accumulate car position

hFileHandle = _lcreat((LPSTR)szFileRst,0);  
if ( hFileHandle != -1 )
{
    fOffRoadDist=0;  
    for( nIndex=0; nIndex<7; nIndex++ )
    {
        nCP[nIndex]=0;  
    }
    for( nIndex=nIgnoreTimes+1;
         nIndex<nRunTimes+1; nIndex++ )
    {
        CarPosition();  
    }
    nOffRoadDist=(long)fOffRoadDist;  
nOffRoadTime=nCP[0]+nCP[6]+nCP[1]+nCP[5];  

    // careate a .RST file and save data

    _llseek( hFileHandle, 0, 0);  
    wssprintf( szDataBuff,
              "%6.6i%6.6i%6.6i%6.6i%6.6i%6.6i%6.6i%3.3i%9.9li%6.6i%6.6i%3.3i%3.3i%6.6i%6.6i%3.3i"
              ,
              nCP[0],nCP[1],nCP[2],nCP[3],nCP[4],
              nCP[5],nCP[6],nObjectShape,
nRBMean00, nRBID, nRBSTD0000, nAvgSpeed000, nETime-800, nITime-850, nOffRoadDist, nOffRoadTime, nDrawCL );

_lwrite( hFileHandle, (LPSTR)szDataBuff, lstrlen(szDataBuff));
_lwrite( hFileHandle, "\r", 1 );
_llseek( hFileHandle, 0, 1);
_lclose( hFileHandle );

break;

default :
return ( DefWindowProc ( hWnd, message, wParam, lParam ) );
}
return ( NULL );

BOOL FAR PASCAL AboutProc ( hWnd, message, wParam, lParam )
HWND hWnd;
UINT message;
UINT wParam;
LONG lParam;
{
switch (message)
{
case WM_INITDIALOG:
return (TRUE);

// if the button OK is pushed, the dialog box will be closed.
case WM_COMMAND:
if (wParam == IDOK)
{
EndDialog(hWnd, TRUE);
return (TRUE);
}
break;
}
return (FALSE);
BOOL FAR PASCAL Dial0Proc ( hWnd, hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    switch ( message )
    {
    case WM_INITDIALOG :
        SetDlgItemText(hDlg, IDD_NAME, (LPSTR)szSubCaption );
        break;

    case WM_COMMAND :
        switch ( wParam )
        {
        case IDOK :
            GetDlgItemText( hWnd, IDD_NAME, (LPSTR)szSubCaption, 25 );
            EndDialog ( hWnd, TRUE );
            break;
        }
        break;

    default:
        return ( FALSE );
    }
    return ( TRUE );
}

BOOL FAR PASCAL Dial1Proc ( hWnd, hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    FARPROC IpFPTProc, IpFPDProc;
    BOOL bFPT, bFPD;
switch ( message )
{
    case WM_INITDIALOG :
        nSeedInit = rand();
        nSeed = nSeedInit;
        SetDlgItemInt ( hDlg, IDD_ROADWIDTH, nRoadWidthft,
                        TRUE ) ;
        SetDlgItemInt ( hDlg, IDD_SEED, nSeed, TRUE );
        SetDlgItemInt ( hDlg, IDD_SEEDSHOW, nSeedtmp, TRUE );
        CheckRadioButton ( hDlg, IDD_FT, IDD_FD,
                             nPreviewSelect );
        if ( nPreviewSelect == IDD_FT )
        {
            SetDlgItemText( hDlg, IDD_PREVIEWSTYLE, "Time" );
            SetDlgItemText( hDlg, IDD_PREVIEWDIST, "XXX" );
            SetDlgItemInt ( hDlg, IDD_INITIALSPEED,
                            nSimultSpeed, TRUE );
            switch ( nPreviewTime )
            {
                case 1402:
                    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "0.20" );
                    break;
                case 1404:
                    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "0.40" );
                    break;
                case 1408:
                    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "0.80" );
                    break;
                case 1410:
                    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "1.00" );
                    break;
                case 1420:
                    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "2.00" );
                    break;
                case 1432:
                    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "3.20" );
                    break;
                case 1450:
                    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "5.00" );
                    break;
                case 1464:
                    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "6.40" );
                    break;
            }
        }
}
break;
}
if( nSpeedVariable == IDD_FIXED )
SetDlgItemText( hDlg, IDD_OBJECTSPEED, "Constant" );
else
SetDlgItemText( hDlg, IDD_OBJECTSPEED, "Variable" );
else{
SetDlgItemText( hDlg, IDD_PREVIEWSTYLE, "Dist" );
SetDlgItemText( hDlg, IDD_PREVIEWTIME, "XXX" );
SetDlgItemInt( hDlg, IDD_INITIALSPEED, nSemSimultSpeed, TRUE );
SetDlgItemInt( hDlg, IDD_PREVIEWDIST, nSemPreDist, TRUE );
if( nSemSpeedVariable == IDD_FIXED )
SetDlgItemText( hDlg, IDD_OBJECTSPEED, "Constant" );
else
SetDlgItemText( hDlg, IDD_OBJECTSPEED, "Variable" );
}
return ( TRUE );

case WM_COMMAND :
switch ( wParam )
{
case IDCANCEL :
EndDialog ( hDlg, FALSE );
break;

case IDOK :
nRoadWidthfttmp = GetDlgItemInt ( hDlg, IDD_ROADWIDTH,
&bBool1, TRUE );
nSeedtmp = GetDlgItemInt ( hDlg, IDD_SEED,
&bBool3, TRUE );
if(nRoadWidthfttmp>50||nRoadWidthfttmp<=18)
{
BWCCMessageBox( hDlg, "Input Error!", "ERROR !", MB_ICONSTOP|MB_OK);
}
else{

nRoadWidthft = nRoadWidthfttmp;
nSeed = nSeedtmp;

switch ( nPreviewSelectTmp )
{
case 0:
    nPreviewSelect = IDD_FT;
    nPreviewTime = nSemPreviewTime;
    nSpeedVariable = nSemSpeedVariable;
    nSimultSpeed = nSemSimultSpeed;
    break;

case 1:
    nPreviewSelect = IDD_FD;
    nPreDist = nSemPreDist;
    nSpeedVariable = nSemSpeedVariable;
    nSimultSpeed = nSemSimultSpeed;
    break;

EndDialog ( hDlg, TRUE );
}
break;

case IDD_FT:
    nPreviewSelectTmp = 0;
    lpFPTProc = MakeProcInstance ((FARPROC)FPTProc, hInst);
    bFPT = DialogBox ( hInst, "FPTbox", hDlg, lpFPTProc );
    FreeProcInstance ( lpFPTProc );
    if ( bFPT == TRUE )
    {
        SetDlgItemText( hDlg, IDD_PREVIEWSTYLE, "Time" );
        SetDlgItemText( hDlg, IDD_PREVIEWDIST, "XXX" );
        SetDlgItemInt ( hDlg, IDD_INITIALSPEED,
                        nSemSimultSpeed, TRUE );
        switch( nSemPreviewTime )
        {
            case 1402:
                SetDlgItemText( hDlg, IDD_PREVIEWTIME, "0.20" );
                break;
            case 1404:
                SetDlgItemText( hDlg, IDD_PREVIEWTIME, "0.40" );
                break;
            case 1408:
                SetDlgItemText( hDlg, IDD_PREVIEWTIME, "0.80" );
                break;
            case 1410:
                SetDlgItemText( hDlg, IDD_PREVIEWTIME, "1.00" );
                break;
        }
case 1420:
    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "2.00" );
    break;

case 1432:
    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "3.20" );
    break;

case 1450:
    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "5.00" );
    break;

case 1464:
    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "6.40" );
    break;
}

else
{
    nSemPreviewTime = nPreviewTime;
    nSemSpeedVariable = nSpeedVariable;
    nSemSimultSpeed = nSimultSpeed;
}
break;

case IDD_FD:
    nPreviewSelectTmp = 1;
lpFPDPProc = MakeProcInstance ((FARPROC)FPDProc, hInst);
bFPD = DialogBox ( hInst, "FPDbox", hDlg, lpFPDPProc);
FreeProcInstance ( lpFPDProc );
if( bFPD == TRUE )
{
    SetDlgItemText( hDlg, IDD_PREVIEWSTYLE, "Dist." );
    SetDlgItemText( hDlg, IDD_PREVIEWTIME, "XXX" );
    SetDlgItemInt ( hDlg, IDD_INITIALSEED, nSemSimultSpeed, TRUE );
    SetDlgItemInt ( hDlg, IDD_PREVIEWDIST, nSemPreDist, TRUE );
    if( nSemSpeedVariable == IDD_FIXED )
SetDlgItemText( hDlg, IDD_OBJECTSPEED, "Constant" );
else
    SetDlgItemText( hDlg, IDD_OBJECTSPEED, "Variable" );
}
else
{
    nSemPreDist = nPreDist;
    nSemSpeedVariable=nSpeedVariable;
    nSemSimultSpeed=nSimultSpeed;
}
break;

case IDD_ROADWIDTH:
    break;

case IDD_SEED:
    break;

    default : return ( FALSE );
}
break;

default:
    return ( FALSE );
}
return ( TRUE );

//=================================
//Dial2Proc()
//=================================
// The dialog box is used to save data.
// Users can input file name in the dialog box.

BOOL FAR PASCAL Dial2Proc ( hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    switch ( message )
    {
    case WM_INITDIALOG :
        DlgDirList( hDlg, "\*.", IDD_DIRECTOR, IDD_PATH, 0x4030 );
        break;
SetDlgItemText ( hDlg, IDD_FILENAME, (LPSTR)szFileName );
return ( TRUE );

case WM_COMMAND :
    switch ( wParam )
    {
        case IDCANCEL :
            EndDialog ( hDlg, FALSE );
            break;

        case IDOK :
            GetDlgItemText ( hDlg, IDD_PATH, (LPSTR)szPathTmp, 69 );
            GetDlgItemText ( hDlg, IDD_FILENAME, (LPSTR)szFileName, 79 );
            EndDialog ( hDlg, TRUE );
            break;

        case IDD_DIRECTOR :
            if( HIWORD( lParam ) == 2 )
            {
                SendDlgItemMessage( hDlg, IDD_DIRECTOR, LB_GETCURSEL, 0, 0L );
                DlgDirSelect( hDlg, (LPSTR)szDirectory, IDD_DIRECTOR );
                lstrcpy((LPSTR)szDirectoryTmp, (LPSTR)szDirectory);
                lstrcat((LPSTR)szDirectoryTmp, "\.*.");
                DlgDirList( hDlg, szDirectoryTmp, IDD_DIRECTOR,
                           IDD_PATH, 0x4030 );
            }
            break;

        default : return ( FALSE );
    }
    break;

default:
    return ( FALSE );
}
return ( TRUE );

// The dialog box allows users to input values of curve
// rate, time lag,
// gain, object shape, and center line.
BOOL FAR PASCAL Dia13Proc ( hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    static HWND hCurves, hLags, hGains, hCtrl;
    static int nCtrlID=901;
    BOOL bBools;
    RECT rRectl;
    HBITMAP hBitmap8;

    switch ( message )
    {
    // initialize the dialog box

    case WM_INITDIALOG:
        CheckRadioButton ( hDlg, IDD_SHAPECAR,
                         IDD_SHAPECIRCLE, nObjectShape );
        CheckRadioButton ( hDlg, IDD_CLYES, IDD_CLNO, nDrawCL );
        hCurves = GetDlgItem(hDlg, IDD_CURVES);
        SetScrollRange ( hCurves, SB_CTL, 0, 100, TRUE );
        SetScrollPos ( hCurves, SB_CTL, nCurves, TRUE );
        SetDlgItemInt ( hDlg, IDD_CURVEE, nCurves, FALSE );

        hLags = GetDlgItem(hDlg, IDD_LAGS);
        SetScrollRange ( hLags, SB_CTL, 0, 1000, TRUE );
        SetScrollPos ( hLags, SB_CTL, nLags, TRUE );
        SetDlgItemInt ( hDlg, IDD_LAGEE, nLags, FALSE );

        hGains = GetDlgItem(hDlg, IDD_GAINS);
        SetScrollRange ( hGains, SB_CTL, 1, 7, TRUE );
        SetScrollPos ( hGains, SB_CTL, nGains, TRUE );
        SetDlgItemInt ( hDlg, IDD_GAINE, nGains, FALSE );
        SetTimer( hDlg, 17, 300, NULL );
        return ( TRUE );

    // delay 300 msec. to laod bitmap of car or circle

    case WM_TIMER:
        KillTimer ( hDlg, 17 );
        SetRect ( &rRectl, 168, 16, 168+33, 16+33 );
        MapDialogRect ( hDlg, &rRectl );
        hDC = GetDC ( hDlg );
        if( nObjectShape==IDD_SHAPECAR )
            hBitmap8 = LoadBitmap ( hInst, "CABITMAP" );
        else
            hBitmap8 = LoadBitmap ( hInst, "CIBITMAP" );
hMemDC = CreateCompatibleDC ( hDC );
SelectObject ( hMemDC, hBitmap8 );
StretchBlt ( hDC, rRect1.left, rRect1.top, rRect1.right-rRect1.left,
          rRect1.bottom-rRect1.top, hMemDC, 0, 0, 75, 75, SRCCOPY );
DeleteDC ( hMemDC );
DeleteObject ( hBitmap8 );
ReleaseDC ( hDlg, hDC );
ValidateRect ( hDlg, NULL );
break;

// set scroll bar

case WM_HSCROLL :
    hCtrl=HIWORD(lParam);
    nCtrlID=GetWindowWord( hCtrl, GWW_ID );
    if( nCtrlID == IDD_CURVES )
    {
        switch ( wParam )
        {
 case SB_PAGEUP:
        nCurves-=10;
        break;

 case SB_PAGEDOWN:
        nCurves+=10;
        break;

 case SB_LINEUP:
        nCurves-=1;
        break;

 case SB_LINEDOWN:
        nCurves+=1;
        break;

 case SB_THUMBPOSITION:
        nCurves=LOWORD(lParam);
        break;
    }
    if ( nCurves>=100)
        nCurves=100;
    if ( nCurves<=0)
        nCurves=0;
    SetScrollPos (hCurves, SB_CTL, nCurves, TRUE);
    SetDlgItemInt( hDlg, IDD_CURVEE, nCurves, TRUE );
    if( nCtrlID == IDD_LAGS)


```c
{
switch ( wParam )
{
  case SB_PAGEUP:
    nLags-=100;
    break;
  
  case SB_PAGEDOWN:
    nLags+=100;
    break;
  
  case SB_LINEUP:
    nLags-=100;
    break;
  
  case SB_LINEDOWN:
    nLags+=100;
    break;
  
  case SB_THUMBPOSITION:
    nLags=(LOWORD(lParam)/100)*100;
    break;
}

if ( nLags>=1000)
    nLags=1000;
if ( nLags<=0)
    nLags=0;
SetScrollPos (hLags, SB_CTL, nLags, TRUE);
SetDlgItemInt( hDlg, IDD_LAGE, nLags, TRUE);
}

if( nCtrlID == IDD_GAINS )
{
switch ( wParam )
{
  case SB_PAGEUP:
    nGains-=1;
    break;
  
  case SB_PAGEDOWN:
    nGains+=1;
    break;
  
  case SB_LINEUP:
    nGains-=1;
    break;
  
  case SB_LINEDOWN:
    nGains+=1;
```
case SB_THUMBPOSITION:
    nGains=LOWORD(lParam);
    break;

    }  // end of if
    if ( nGains>=7)
        nGains=7;
    if ( nGains<=0)
        nGains=0;
    SetScrollPos (hGains, SB_CTL, nGains, TRUE);
    SetDlgItemInt( hDlg, IDD_GAINE, nGains, TRUE);
    break;

case WM_COMMAND :
    switch ( wParam )
    {
    case IDCANCEL :
        EndDialog( hDlg, FALSE );
        break;
    case IDOK
        if( nCurves>100 || nCurves<0 || nGains>7 || nGains<0 ||
            nLags>1000 || nLags<0)
        {
            BWCCMessageBox( hDlg, "Input Error!", "ERROR!",
            MB_ICONSTOP|MB_OK);
        }
        else
        {
            switch(nShapeTmp)
            {
                case 0:
                    nObjectShape = IDD_SHAPECAR;
                    break;

                case 1:
                    nObjectShape = IDD_SHAPECIRCLE;
                    break;
            }
            switch(nCLTmp)
            {
                case 0:
                    nDrawCL = IDD_CLNO;
                    break;
                }
case 1:
    nDrawCL = IDD_CLYES;
    break;
}
EndDialog ( hDlg, TRUE );
break;

case IDD_CURVEE :
    nCurves=GetDlgItemInt ( hDlg, IDD_CURVEE, &bBools, TRUE);
    hCurves=GetDlgItem( hDlg, IDD_CURVES);
    SetScrollPos (hCurves, SB_CTL, nCurves, TRUE );
    break;

case IDD_LAGE :
    nLags=GetDlgItemInt( hDlg, IDD_LAGE, &bBools, TRUE);
    hLags=GetDlgItem( hDlg, IDD_LAGS);
    SetScrollPos (hLags, SB_CTL, nLags, TRUE );
    break;

case IDD_GAINE :
    nGains=GetDlgItemInt( hDlg, IDD_GAINE, &bBools, TRUE);
    hGains=GetDlgItem( hDlg, IDD_GAINS);
    SetScrollPos (hGains, SB_CTL, nGains, TRUE );
    break;

case IDD_CLYES :
    nCLTmp = 1;
    break;

case IDD_CLNO :
    nCLTmp = 0;
    break;

case IDD_SHAPECAR :
    nShapeTmp = 0;
    SetRect ( &rRect1, 168, 16, 168+33, 16+33 );
    MapDialogRect ( hDlg, &rRect1 );
    hDC = GetDC ( hDlg );
    hBitmap8 = LoadBitmap ( hInst, "CABITMAP" );
    hMemDC = CreateCompatibleDC ( hDC );
    SelectObject ( hMemDC, hBitmap8 );
    StretchBlt ( hDC, rRect1.left, rRect1.top, rRect1.right-rRect1.left,
                 rRect1.bottom-rRect1.top, hMemDC, 0, 0, 75, 75, SRCCOPY );
DeleteDC ( hMemDC );
DeleteObject ( hBitmap8 );
ReleaseDC ( hDlg, hDC );
ValidateRect ( hDlg, NULL );
break;

case IDD_SHAPECIRCLE :
nShapeTmp = 1;
SetRect ( &rRectl, 168, 16, 168+33, 16+33 );
MapDialogRect ( hDlg, &rRectl );
hDC = GetDC ( hDlg );
hBitmap8 = LoadBitmap ( hInst, "CIBITMAP" );
hMemDC = CreateCompatibleDC ( hDC );
SelectObject ( hMemDC, hBitmap8 );
StretchBlt ( hDC, rRectl.left, rRectl.top, 
   rRectl.right-rRectl.left, 
   rRectl.bottom-rRectl.top, hMemDC, 
   0, 0, 75, 75, SRCCOPY );
DeleteDC ( hMemDC );
DeleteObject ( hBitmap8 );
ReleaseDC ( hDlg, hDC );
ValidateRect ( hDlg, NULL );
break;

default : return ( FALSE );
}
break;

default:
   return ( FALSE );
}
return ( TRUE );

햔ационный код:

The dialog box allows users to input values of experiment time, subject adapt time, and first curve direction.

BOOL FAR PASCAL Dia14Proc ( hWnd hDlg, uMessage, wParam, lParam )
{ 
   switch ( message ) 
   { 
      case WM_INITDIALOG :
         CheckRadioButton ( hDlg, IDD_1, IDD_5, nETime );
         CheckRadioButton ( hDlg, IDD_05, IDD_30, nITime );
   }
CheckRadioButton ( hDlg, IDD_RIGHTCURVE, IDD_LEFTCURVE, nCurveDir );
return ( TRUE );

case WM_COMMAND :
    switch ( wParam )
    {
    case IDCANCEL :
        EndDialog ( hDlg, FALSE );
        break;
    case IDOK :
        switch(nETimeTmp) {
            case 1:
                nETime = IDD_1;
                break;
            case 2:
                nETime = IDD_2;
                break;
            case 3:
                nETime = IDD_3;
                break;
            case 4:
                nETime = IDD_4;
                break;
            case 5:
                nETime = IDD_5;
                break;
        }
        switch(nITimeTmp) {
            case 1:
                nITime = IDD_05;
                break;
            case 2:
                nITime = IDD_10;
                break;
            case 3:
                nITime = IDD_15;
                break;
        }
case 4:
nITime = IDD_20;
break;

case 5:
nITime = IDD_30;
break;
}
switch(nCurveDirTmp)
{
    case 0:
nCurveDir = IDD_RIGHTCURVE;
    break;

    case 1:
nCurveDir = IDD_LEFTCURVE;
    break;
}

EndDialog ( hDlg, TRUE );
break;

case IDD_1 :
nETimeTmp = 1;
break;

case IDD_2 :
nETimeTmp = 2;
break;

case IDD_3 :
nETimeTmp = 3;
break;

case IDD_4 :
nETimeTmp = 4;
break;

case IDD_5 :
nETimeTmp = 5;
break;

case IDD_05 :
nITimeTmp = 1;
break;

case IDD_10 :
nITimeTmp = 2;
break;
case IDD_15 :
    nITimeTmp = 3;
    break;

case IDD_20 :
    nITimeTmp = 4;
    break;

case IDD_30 :
    nITimeTmp = 5;
    break;

case IDD_RIGHTCURVE :
    nCurveDirTmp = 0;
    break;

case IDD_LEFTCURVE :
    nCurveDirTmp = 1;
    break;

default : return ( FALSE );
}
break;

default:
    return ( FALSE );
}
return ( TRUE );

BOOL FAR PASCAL FPDProc ( hWnd hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    static HWND hCtrl, hPD, hIS;
    static int nCtrlID=1301;
    BOOL bBools;

    switch ( message )
    {
    case WM_INITDIALOG :
nSemPreDist = nPreDist;
nSemSpeedVariable=nSpeedVariable;
nSemSimultSpeed=nSimultSpeed;

CheckRadioButton ( hDlg, IDD_FIXED, IDD_VARIABLE, 
    nSemSpeedVariable );

hPD=GetDlgItem( hDlg, IDD_PDSB);
SetScrollRange (hPD, SB_CTL, 0, 90, TRUE);
SetScrollPos (hPD, SB_CTL, nSemPreDist, TRUE );
SetDlgItemInt ( hDlg, IDD_PD, nSemPreDist, TRUE );

hIS=GetDlgItem( hDlg, IDD_SPEEDSB);
SetDlgItemInt ( hDlg, IDD_LS, 0, TRUE );
SetDlgItemInt ( hDlg, IDD_HS, 120, TRUE );
SetScrollRange (hIS, SB_CTL, 0, 120, TRUE);
SetScrollPos (hIS, SB_CTL, nSemSimultSpeed, TRUE);
SetDlgItemInt ( hDlg, IDD_IS, nSemSimultSpeed, TRUE );

return ( TRUE );

case WM_HSCROLL :
    hCtrl=HIWORD(lParam);
nCtrlID=GetWindowWord( hCtrl, GWW_ID );
    if( nCtrlID == IDD_PDSB )
    {
        switch( wParam )
        {
            case SB_PAGEUP:
                nSemPreDist-=5;
                break;

            case SB_PAGEDOWN:
                nSemPreDist+=5;
                break;

            case SB_LINEUP:
                nSemPreDist-=1;
                break;

            case SB_LINEDOWN:
                nSemPreDist+=1;
                break;

            case SB_THUMBPOSITION:
                nSemPreDist=LOWORD(lParam);
                break;
        }
    }
if ( nSemPreDist>90 )
    nSemPreDist=90;
if ( nSemPreDist<=0 )
    nSemPreDist=0;
SetScrollPos (hPD, SB_CTL, nSemPreDist, TRUE);
SetDlgItemInt ( hDlg, IDD_PD, nSemPreDist, TRUE );
}
   if( nCtrlID == IDD_SPEEDSB)
   {
   switch( wParam )
   {
   case SB_PAGEUP:
       nSemSimultSpeed-=5;
       break;
   case SB_PAGEDOWN:
       nSemSimultSpeed+=5;
       break;
   case SB_LINEUP:
       nSemSimultSpeed-=1;
       break;
   case SB_LINEDOWN:
       nSemSimultSpeed+=1;
       break;
   case SB_THUMBPOSITION:
       nSemSimultSpeed=LOWORD(lParam);
       break;
   }
   if ( nSemSimultSpeed>=120)
       nSemSimultSpeed=120;
   if ( nSemSimultSpeed<=0)
       nSemSimultSpeed=0;
   hIS=GetDlgItem( hDlg, IDD_SPEEDSB);
   SetScrollPos (hIS, SB_CTL, nSemSimultSpeed, TRUE);
   SetDlgItemInt ( hDlg, IDD_IS, nSemSimultSpeed, TRUE );
}
   break;

case WM_COMMAND :
    switch ( wParam )
    {
    case IDCANCEL :
       EndDialog ( hDlg, FALSE );
       break;
case IDOK :
    if(
        nSemPresDist>90||nSemPresDist<0||nSemSimultSpeed>120||nSemSimultSpeed<0)
    {
        BWCCMessageBox( hDlg, "Input Error!", "ERROR!",
            MB_ICONSTOP|MB_OK);
    } else
    {
        switch ( nSpeedVariableTmp )
        {
            case 0:
                nSemSpeedVariable = IDD_FIXED;
                break;

            case 1:
                nSemSpeedVariable = IDD_VARIABLE;
                break;
        }
        EndDialog ( hDlg, TRUE );
    } break;

case IDD_FIXED:
    nSpeedVariableTmp = 0;
    break;

case IDD_VARIABLE:
    nSpeedVariableTmp = 1;
    break;

    default : return ( FALSE );
    } break;

default:
    return ( FALSE );
}
return ( TRUE );

//=----------------------------------------------------------------------------------
//= FPTProc()
//=----------------------------------------------------------------------------------
// If the experiment selects the constant preview time,
// the dialog will be presented for setting variables.

BOOL FAR PASCAL FPTProc ( hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    static HWND hCtrl, hIS;
    static int nCtrl1ID=1301;
    BOOL bBools;

    switch (message)
    {
        case WM_INITDIALOG:
            nSemPreviewTime = nPreviewTime;
            nSemSpeedVariable = nSpeedVariable;
            nSemSimultSpeed = nSimultSpeed;

            CheckRadioButton ( hDlg, IDD_FIXED, IDD_VARIABLE, nSemSpeedVariable );
            CheckRadioButton ( hDlg, IDD_020, IDD_640, nSemPreviewTime );

            switch( nSemPreviewTime )
            {
            case 1402:
                nPreviewTimeTmp = 0;
                break;

            case 1404:
                nPreviewTimeTmp = 1;
                break;

            case 1408:
                nPreviewTimeTmp = 2;
                break;

            case 1410:
                nPreviewTimeTmp = 3;
                break;

            case 1420:
                nPreviewTimeTmp = 4;
                break;

            case 1432:
                nPreviewTimeTmp = 5;
                break;

            case 1450:
            
    }
nPreviewTimeTmp = 6;
break;

case 1464:
nPreviewTimeTmp = 7;
break;
}

if( nSemSimultSpeed >= nHSpeed[nPreviewTimeTmp] )
nSemSimultSpeed = nHSpeed[nPreviewTimeTmp];

hIS=GetDlgItem( hDlg, IDD_SPEEDSB);
SetDlgItemInt ( hDlg, IDD_LS, 0, TRUE );
SetDlgItemInt ( hDlg, IDD_Hs,
  nHSpeed[nPreviewTimeTmp], TRUE );
SetScrollRange (hIS, SB_CTL, 0,
  nHSpeed[nPreviewTimeTmp], TRUE);
SetScrollPos (hIS, SB_CTL, nSemSimultSpeed, TRUE);
SetDlgItemInt(hDlg, IDD_IS, nSemSimultSpeed, TRUE );
return ( TRUE );

case WM_HSCROLL :
  hCtrl=HIWORD(lParam);
nCtrlID=GetWindowWord( hCtrl, GWW_ID );
  if( nCtrlID == IDD_SPEEDSB )
  {
    switch( wParam )
    {
      case SB_PAGEUP:
nSemSimultSpeed-=5;
      break;

      case SB_PAGEDOWN:nSemSimultSpeed+=5;
      break;

      case SB_LINEUP:nSemSimultSpeed-=1;
      break;

      case SB_LINEDOWN:nSemSimultSpeed+=1;
      break;

      case SB_THUMBPOSITION:
nSemSimultSpeed=LOWORD(lParam);
      break;
    }
if ( nSemSimultSpeed>=nHSpeed[nPreviewTimeTmp])
    nSemSimultSpeed=nHSpeed[nPreviewTimeTmp];
if ( nSemSimultSpeed<=0)
    nSemSimultSpeed=0;
SetScrollPos (hIS, SB_CTL, nSemSimultSpeed, TRUE);
SetDlgItemInt ( hDlg, IDD_IS, nSemSimultSpeed, TRUE );
}
break;
case WM_COMMAND :
    switch ( wParam )
    {
    case IDCANCEL :
        EndDialog ( hDlg, FALSE );
        break;
    case IDOK :
        if( nSemSimultSpeed>nHSpeed[nPreviewTimeTmp]
            ||nSemSimultSpeed<0)
        {
            BWCCMessageBox( hDlg, "Input Error!", "ERROR !", MB_ICONSTOP|MB_OK);
        }
        else
        {
            switch ( nSpeedVariableTmp )
            {
            case 0:
                nSemSpeedVariable = IDD_FIXED;
                break;
            case 1:
                nSemSpeedVariable = IDD_VARIABLE;
                break;
            }
            switch( nPreviewTimeTmp )
            {
            case 0:
                nSemPreviewTime = 1402;
                break;
            case 1:
                nSemPreviewTime = 1404;
                break;
            case 2:
                nSemPreviewTime = 1408;
            }
The Source Code of TRACER (TRACER.C)

```
#include <windows.h> // header files, from the
#include <stdlib.h>  // built-in libraries
#include <string.h>  // of C compiler
#include <stdio.h>
#include <math.h>
#include "tracer.h"   // user-defined header file

// Function definitions
long FAR PASCAL FrameWndProc (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL CloseEnumProc (HWND, LONG);
long FAR PASCAL TextWndProc (HWND, UINT, UINT, LONG);
long FAR PASCAL GraphWndProc (HWND, UINT, UINT, LONG);
long FAR PASCAL InfoWndProc (HWND, UINT, UINT, LONG);
long FAR PASCAL RstWndProc (HWND, UINT, UINT, LONG);
long FAR PASCAL CRWndProc (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL AboutProc (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL Dial1Proc (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL Dial2Proc (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL Dial3Proc (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL Dial4Proc (HWND, UINT, UINT, LONG);
BOOL FAR PASCAL Dial5Proc ( HWND, UINT, UINT, LONG);
int FAR PASCAL AbortProc (HDC, int);
int FAR PASCAL PrintDlgProc (HWND, UINT, UINT, LONG);
in t GetValue(int);
H DC GetPrinterDC(void);

// defining structure for storing data unique to each
// child window

t ypedef struct
{
    int nSubIndex;
    int nRunTimes;
    int nETime;
    int nSEG;
    int nLR[602];
    int nCP[602];
    int nCS[602];
    int nScrollValue;
    int nhScrollValue;
    int nRWft;
    int nOW;
} DATABUFFER;
```
typedef DATABUFFER NEAR *NPDATABUFFER;

typedef struct
{
    int nBuff[15];
    char szSd[10];
    char szSID[20];
} INFOBUFFER;

typedef INFOBUFFER NEAR *NPINFOBUFFER;

typedef struct
{
    int nRate[9];
    int nOS;
    int nRBID;
    int nRBMeanI, nRBMeanF;
    int nRBSTDI;
    int nRBSTDF;
    int nAvgSpeedI;
    int nAvgSpeedF;
    int nETime;
    int nITime;
    int nOffRoadDist1, nOffRoadDist2, nOffRoadTime;
    int nDrawCL;
} RSTBUFFER;

typedef RSTBUFFER NEAR *NPRSTBUFFER;

// window classes
char szFrameClass[] = "TRACER";
char szTextClass[] = "TextDisplay";
char szGraphClass[] = "GraphDisplay";
char szInfoClass[] = "InfoDisplay";
char szRstClass[] = "ResultDisplay";
char szCRClass[] = "CRClass";

// window title
char szCRTitle[] = "About TRACER 1.0";

// general output buffer
char szDataBuff[50];
char szTextOutput[100];

// File name, Path name
char szFileName[80], szFileExtention[90], szDirectory[60];
char szDirectoryTmp[60];
char szPathTmp[60], szCaption[100];

// the height and width of the window
int sHeight, sWidth;

// handles
HANDLE hInst;
HMENU hMainMenu;
HMENU hSubMenu;

// pre defined data array to store experiment data
int nLeftRoad[3001], nCarPos[3001], nCarSpeed[3001];

// general index variables
int nIndex;
int nDisplayMode = 910;
int nDisplayModeTemp;
int hFileHandle;

// templey buffers
char szl00[1], sz10[1], sz1[1], sz[9];
int n100, n10, n1, n[9];

// structure of rectangle and bitmaps
RECT rRect;
HBITMAP hBitmap, hOldBitmap;

// the road width
int nRWft = 50;

// the object width
int nOW = 29;

// window handles
HWND hWndClient, hWndFrame;

// experimental variables
int nPreDist, nRoadWidthft, nSimultSpeed, nSpeedVariable;
int nCurves, nLags, nGains, nObjectShape, nDrawCL;

// clear the contents in output buffers
char szSubjectID[20] = "";
char szSubjectIDtmp[26] = "";
char szSeed[10] = "";
char szSeedtmp[12] = "";

// 7 car positions
int nCP[7];
float fCP[7];
long int nRCSTD00000;

int nRBMean00I, nRBMean00F;
int nRBID, nRCID, nRBSTD1, nRBSTDF, nAvgSpeedI, nAvgSpeedF;

// integer buffer
int n57, n60, n63;

// variables for printing job
HDC hPrintDC;
HWND hPrintDlg;
BOOL bUserCancels;

// variables for Dialog box 4
int nETimei; // Experiment time
int nRunTimes; // how many iterations in experiment time
int nITimei; // Subject adapt time
int nIgnoreTimes;
int nCurveDir;
int nPreviewSelect, nPreviewTime;
int nOffRoadDist1, nOffRoadDist2, nOffRoadTime;

// variables for Dial1
int nSEG=701;
int nSEGTmp=1;
int nSEGBUFFER = 701;
int nSubIndex, nIndexTmp;

// main function of this program
//====================================================================================================
// WinMain()
//====================================================================================================
int PASCAL WinMain (HANDLE hlnstance, HANDLE hPrevlnstance, LPSTR lpszCmdLine, int nCmdShow)
{
// auto variables
HANDLE hAccel; // for accelerate keys
HWND hWndFrame, hWndClient, hCRWnd;
MSG msg;
WNDCLASS WndClass;
sWidth = GetSystemMetrics ( SM_CXSCREEN );
sHeight = GetSystemMetrics ( SM_CYSCREEN );
hInst = hlnstance;

// initializing a new window
if (!hPrevlnstance)
{
    // Register the frame window Class
WndClass.style = CS_HREDRAW | CS_VREDRAW;
WndClass.lpfnWndProc = FrameWndProc;
WndClass.cbClsExtra = 0;
WndClass.cbWndExtra = 0;
WndClass.hInstance = hInstance;
WndClass.hIcon = LoadIcon (hInstance, "TRACERICON");
WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
WndClass.hbrBackground = COLOR_APPWORKSPACE + 1;
WndClass.lpszMenuName = NULL;
WndClass.lpszClassName = szFrameClass;

RegisterClass (&WndClass);

// Register the Copyright child window Class

WndClass.style = CS_HREDRAW | CS_VREDRAW;
WndClass.lpfnWndProc = CRWndProc;
WndClass.cbClsExtra = 0;
WndClass.cbWndExtra = 0;
WndClass.hInstance = hInstanceId;
WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
WndClass.lpszMenuName = NULL;
WndClass.hIcon = NULL;
WndClass.lpszClassName = szCRClass;
WndClass.hbrBackground = GetStockObject(WHITE_BRUSH);

RegisterClass (&WndClass);

// Register the Text child window Class

WndClass.style = CS_HREDRAW | CS_VREDRAW;
WndClass.lpfnWndProc = TextWndProc;
WndClass.cbClsExtra = 0;
WndClass.cbWndExtra = sizeof (LOCALHANDLE);
WndClass.hInstance = hInstance;
WndClass.hIcon = LoadIcon(NULL, IDI_APPLICATION);
WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
WndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
WndClass.lpszMenuName = NULL;
WndClass.lpszClassName = szTextClass;

RegisterClass (&WndClass);

// Register the Graph child window Class

WndClass.style = CS_HREDRAW | CS_VREDRAW;
WndClass.lpfnWndProc = GraphWndProc;
WndClass.cbClsExtra = 0;
WndClass.cbWndExtra = sizeof (LOCALHANDLE);
WndClass.hInstance = hInstance;
WndClass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
WndClass.hCursor = LoadCursor (NULL, IDC_ARROW);
WndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
WndClass.lpszMenuName = NULL;
WndClass.lpszClassName = szGraphClass;

RegisterClass (&WndClass);

// Register the information child window Class

WndClass.style = CS_HREDRAW | CS_VREDRAW;
WndClass.lpfnWndProc = InfoWndProc;
WndClass.cbClsExtra = 0;
WndClass.cbWndExtra = sizeof (LOCALHANDLE);
WndClass.hInstance = hInstance;
WndClass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
WndClass.hCursor = LoadCursor (NULL, IDC_ARROW);
WndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
WndClass.lpszMenuName = NULL;
WndClass.lpszClassName = szInfoClass;

RegisterClass (&WndClass);

// Register the result child window Class

WndClass.style = CS_HREDRAW | CS_VREDRAW;
WndClass.lpfnWndProc = RstWndProc;
WndClass.cbClsExtra = 0;
WndClass.cbWndExtra = sizeof (LOCALHANDLE);
WndClass.hInstance = hInstance;
WndClass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
WndClass.hCursor = LoadCursor(NULL, IDC_ARROW);
WndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
WndClass.lpszMenuName = NULL;
WndClass.lpszClassName = szRstClass;

RegisterClass (&WndClass);

// Obtain handles to three possible menus & submenus
hMainMenu = LoadMenu (hInst, "MainMenu");
hSubMenu = GetSubMenu (hMainMenu, 0);
// Load accelerator table
hAccel = LoadAccelerators(hInst, "MenuAccel");

// Create the frame window

hWndFrame = CreateWindow(szFrameClass, "TRACER 1.0", WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN, 0, 0, sWidth, sHeight, NULL, hMainMenu, hInstance, NULL);

hWndClient = GetWindow(hWndFrame, GW_CHILD);
ShowWindow(hWndFrame, nCmdShow);
UpdateWindow(hWndFrame);

// Enter the message loop
while (GetMessage(&msg, NULL, 0, 0)) {
    if (!TranslateMDISysAccel(hWndClient, &msg) && !TranslateAccelerator(hWndFrame, hAccel, &msg)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
}

// Clean up by deleting unattached menus
return msg.wParam;

// the function to create the main window. (Frame window)
//=====================================================
//=====================================================
long FAR PASCAL FrameWndProc(HWND hWnd, UINT message, UINT wParam, LONG lParam)
{
    CLIENTCREATESTRUCT clientcreate;

    // Far procedure for dialog box procedures
    FARPROC lpfnEnum, lpDial1Proc, lpDial2Proc;
    FARPROC lpDial3Proc, lpDial4Proc, lpDial5Proc, lpAboutProc;
    HWND hWndChild, hWndTarget, hCRWnd;
    MDICREATESTRUCT mdc;create;
    BOOL bData1, bDial1, bDial2, bDial3,
    bDial4, bDial5;

    // proceeding the windows message
    switch (message) {
        case WM_CREATE:
// Create the client window
clientcreate.hWindowMenu = hSubMenu;
clientcreate.idFirstChild = IDM_FIRSTCHILD;
hWndClient = CreateWindow ("MDICLIENT", NULL,
WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
0, 0, 0, 0, hWnd, 1, hInst,
(LPSTR) &clientcreate);

// create the copyright window
mdicreate.szClass = szCRClass;
mdicreate.szTitle = szCRTitle;
mdicreate.hOwner = hInst;
mdicreate.x = 0;
mdicreate.y = 0;
mdicreate.cx = sWidth;
mdicreate.cy = sHeight;
mdicreate.style = 0;
mdicreate.lParam = NULL;

hWndChild = (HWND) SendMessage
(hWndClient, WM_MDICREATE, 0,
(long)(LPMDICREATESTRUCT) &mdicreate);

return 0;

case WM_COMMAND:
    switch (wParam)
    {
    case IDM_ABOUT:
        lpAboutProc = MakeProcInstance((FARPROC)
        AboutProc, hInst);
        DialogBox(hInst,"Aboutbox", hWnd,
        lpAboutProc);
        FreeProcInstance(lpAboutProc);
        break;

    case IDM_OPENDAT:

        // Create a series dialog boxes to collect
        // data
        lpDial12Proc = MakeProcInstance((FARPROC)
        Dial2Proc, hInst);
        bDial2 = DialogBox(hInst, "Dial2", hWnd,
        lpDial12Proc);
        FreeProcInstance(lpDial12Proc);
        if( bDial2 == IDOK )
lpDial3Proc = MakeProcInstance(( FARPROC )Dia13Proc, hInst );
bDial3 = DialogBox( hInst, "Dial3", hWnd, lpDial3Proc );
FreeProcInstance( lpDial3Proc );
if( bDial3 == IDOK )
{
    // TO get file name
    lstrcpy((LPSTR)szFileExtention, (LPSTR)szPathTmp );
    if((int)szPathTmp[3]!=O)
        lstrcat((LPSTR)szFileExtention, (LPSTR)"\" );
        lstrcat((LPSTR)szFileExtention, (LPSTR)szFileName );

    hFileHandle=_lopen((LPSTR)szFileExtention,0);
    if ( hFileHandle != -1 )
    {
        // read data from .data file
        nETime = GetValue(0)+800;
        nITime = GetValue(3)+850;
        nRWft = GetValue(6);
        nOW = GetValue(9);
        switch( nETime )
        {
            case 801:
                nRunTimes=600;
                break;
            case 802:
                nRunTimes=1200;
                break;
            case 803:
                nRunTimes=1800;
                break;
            case 804:
                nRunTimes=2400;
                break;
            case 805:
                nRunTimes=3000;
                break;
        }
        switch( nITime )
        {
            case 801:
                nRunTimes=600;
                break;
            case 802:
                nRunTimes=1200;
                break;
            case 803:
                nRunTimes=1800;
                break;
            case 804:
                nRunTimes=2400;
                break;
            case 805:
                nRunTimes=3000;
                break;
        }
    }
{  
    case 851:  
        nIgnoreTimes=50;  
        break;  
    case 852:  
        nIgnoreTimes=100;  
        break;  
    case 853:  
        nIgnoreTimes=150;  
        break;  
    case 854:  
        nIgnoreTimes=200;  
        break;  
    case 855:  
        nIgnoreTimes=300;  
        break;  
}

lpDial1Proc = MakeProcInstance(( FARPROC )Dial1Proc, hInst );
bDial1 = DialogBox( hInst, "Dial1", hWnd,  
    lpDial1Proc );
FreeProcInstance( lpDial1Proc );
if( bDial1 == IDOK )  
{
    if( nSEGBUFFER == 700 )  
    {  // read all segments

        // If memory is not enough to create so many  
        // child windows, program will be closed.

        bDatal = BWCCMessageBox( hWnd,  
            "Don't open too many child windows  
            at one time. They will exhaust memory.",  
            "Warning!", MB_ICONINFORMATION |  
            MB_OKCANCEL );

        if( bDatal==IDOK)  
            {  
                for( nSEG = 701; nSEG < 701+nETime-800;  
                    nSEG++)  
                    {  
                        if ( nDisplayMode == 910 )  
                            {  

lstrcpy((LPSTR) szCaption, (LPSTR)szFileName);
wsprintf(szTextOutput, " : [%1.1f]", nSEG-700);
lstrcat((LPSTR) szCaption, (LPSTR)szTextOutput);
lstrcat((LPSTR) szCaption," : [Text]" );
mdicreate.szClass = szTextClass;
mdicreate.szTitle = szCaption ; //"Text Display" ;
mdicreate.hOwner = hInst ;
mdicreate.x = CW_USEDEFAULT ;
mdicreate.y = CW_USEDEFAULT ;
mdicreate.cx = CW_USEDEFAULT ;
mdicreate.cy = CW_USEDEFAULT ;
mdicreate.style = WS_VSCROLL ;
mdicreate.lParam = NULL ;

hWndChild = (HWND)SendMessage
    (hWndClient, WM_MDICREATE, 0,
    (long)(LPMDICREATESTRUCT)&mdicreate) ;}
else
{
lstrcpy((LPSTR) szCaption, (LPSTR)szFileName);
wsprintf(szTextOutput, " : [%1.1f]", nSEG-700);
lstrcat((LPSTR) szCaption, (LPSTR)szTextOutput);
lstrcat((LPSTR) szCaption, " : [Text]" );
mdicreate.szClass = szTextClass;
mdicreate.szTitle = szCaption ; //"Text Display" ;
mdicreate.hOwner = hInst ;
mdicreate.x = CW_USEDEFAULT ;
mdicreate.y = CW_USEDEFAULT ;
mdicreate.cx = CW_USEDEFAULT ;
mdicreate.cy = CW_USEDEFAULT ;
mdicreate.style = WS_VSCROLL ;
mdicreate.lParam = NULL ;

hWndChild = (HWND)SendMessage(hWndClient, WM_MDICREATE, 0,
    (long)(LPMDICREATESTRUCT)&mdicreate) ;
} } } } }
else
{
if ( nDisplayMode == 910 )
{
    lstrcpy( (LPSTR) szCaption,
             (LPSTR)szFileName );
    wsprintf(szTextOutput, " : [%1.1i]", nSEG-700);
    lstrcat( (LPSTR) szCaption,
             (LPSTR)szTextOutput );
    lstrcat( (LPSTR) szCaption, " : [Text] " );

    // create a text display window
    mdicreate.szClass = szTextClass ;
    mdicreate.szTitle = szCaption ;
    mdicreate.hOwner = hInst ;
    mdicreate.x = CW_USEDEFAULT ;
    mdicreate.y = CW_USEDEFAULT ;
    mdicreate.cx = CW_USEDEFAULT ;
    mdicreate.cy = CW_USEDEFAULT ;
    mdicreate.style = WS_VSCROLL ;
    mdicreate.lParam = NULL ;

    hWndChild = (HWND) SendMessage ( hWndClient, 
                        WM_MDICREATE, 0, 
                        (long) (LPMDICREATESTRUCT) &mdicreate ) ;
}
else
{
    lstrcpy( (LPSTR) szCaption,
             (LPSTR)szFileName );
    wsprintf( szTextOutput, " : [%1.1i]", nSEG-700);
    lstrcat( (LPSTR) szCaption,
             (LPSTR)szTextOutput );
    lstrcat( (LPSTR) szCaption, " : [Graphic] " );

    mdicreate.szClass = szGraphClass ;
    mdicreate.szTitle = szCaption ;
    mdicreate.hOwner = hInst ;
    mdicreate.x = CW_USEDEFAULT ;
    mdicreate.y = CW_USEDEFAULT ;
    mdicreate.cx = CW_USEDEFAULT ;
    mdicreate.cy = CW_USEDEFAULT ;
    mdicreate.style = WS_VSCROLL | 
                      WS_HSCROLL ;
    mdicreate.lParam = NULL ;

    hWndChild = (HWND) SendMessage ( hWndClient, 

WM_MDICREATE, 0, 
    (long) (LPMDICREATESTRUCT) &mdicreate) ;
} } } } )
return 0 ;

case IDM_OPENINF:
    
    // look for the *.inf file
    lpDial4Proc = MakeProcInstance(( FARPROC )Dial4Proc, hInst );
    bDial4 = DialogBox( hInst, "Dial2", hWnd, 
        lpDial4Proc );
    FreeProcInstance( lpDial4Proc );
    if( bDial4 == IDOK )
    { 
        lstrcpy((LPSTR)szFileExtention, 
            (LPSTR)szPathTmp );
        if((int)szPathTmp[3]!=0)
            lstrcat((LPSTR)szFileExtention,(LPSTR)"\" ");
        lstrcat((LPSTR)szFileExtention, 
            (LPSTR)szFileName );

        hFileHandle=_lopen((LPSTR)szFileExtention,0);
        if ( hFileHandle != -1 )
        {
            
            // read data from disks & reformate data
            nPreDist = GetValue( 0 );
            nRoadWidthft = GetValue( 3 );
            nSimultSpeed = GetValue( 6 );
            nSpeedVariable = GetValue( 9 );
            nCurves = GetValue( 21 );
            nLags = GetValue(24)*1000+GetValue( 27 );
            nGains = GetValue( 30 );
            nObjectShape = GetValue( 33 );
            nDrawCL= GetValue( 36 );
            nETime = GetValue( 39 )+800;
            nITime = GetValue( 42 )+850;
            nCurveDir= GetValue( 45 );
            nPreviewSelect = GetValue( 48 )+1100;
            nPreviewTime = GetValue( 51 )+1400;

            lstrcpy((LPSTR)szSeed, (LPSTR)"" );
            for( nIndex=12; nIndex<21; nIndex++ )
            {
                _llseek( hFileHandle, nIndex, 0 );
                _lread( hFileHandle, (LPSTR)sz1, 1 );
                szSeed[nIndex-12]=*sz1;
lstrcpy((LPSTR)szSubjectID, (LPSTR)"");
for( nIndex=54; nIndex<74; nIndex++)
{
    _llseek( hFileHandle, nIndex, 0);
    _lread( hFileHandle, (LPSTR)szl, 1);
    szSubjectID[nIndex-54] = *szl;
}
_lclose(hFileHandle);

lstrcpy((LPSTR)szCaption, (LPSTR)szFileName);
lstrcat((LPSTR)szCaption, " : [Info] ");
mdicreate.szClass = szInfoClass;
mdicreate.szTitle = szCaption;
mdicreate.hOwner = hInst;
mdicreate.x = CW_USEDEFAULT;
mdicreate.y = CW_USEDEFAULT;
mdicreate.cx = CW_USEDEFAULT;
mdicreate.cy = CW_USEDEFAULT;
mdicreate.style = 0;
mdicreate.lParam = NULL;

hWndChild = (HWND)SendMessage(hWndClient, WM_MDICREATE, 0, (long)(LPMDICREATESTRUCT)&mdicreate);
return 0;

case IDM_OPENRST:

    // look for the *.rst file
    lpDial5Proc = MakeProcInstance((FARPROC)Dial5Proc, hInst);
bDial5 = DialogBox(hInst, "Dial2", hWnd, lpDial5Proc);
FreeProcInstance(lpDial5Proc);
if( bDial5 == IDOK )
{
    lstrcpy((LPSTR)szFileExtention, (LPSTR)szPathTmp);
    if((int)szPathTmp[3]!=0)
lstrcat((LPSTR)szFileExtention, (LPSTR)"
    ");
lstrcat((LPSTR)szFileExtention, (LPSTR)szFileName);
hFileHandle=_lopen
    ((LPSTR)szFileExtention,0);
if ( hFileHandle != -1 )
{

    // read data from disks
    nCP[0] = GetValue( 0 )*1000+GetValue( 3 );
    nCP[1] = GetValue( 6 )*1000+GetValue( 9 );
    nCP[2] = GetValue(12)*1000+GetValue( 15 );
    nCP[3] = GetValue(18)*1000+GetValue( 21 );
    nCP[4] = GetValue(24)*1000+GetValue( 27 );
    nCP[5] = GetValue(30)*1000+GetValue( 33 );
    nCP[6] = GetValue(36)*1000+GetValue( 39 );
    nObjectShape = GetValue( 45 );
    nRBMeanO0I=GetValue(48);
    nRBMeanO0F=GetValue(51);
    nRBID = GetValue( 54 );
    n57 = GetValue(57);
    n60 = GetValue(60);
    n63 = GetValue(63);

    nRBSTDI=100*n57+n60/10;
    nRBSTDF=(n60%10)*1000+n63;
    nAvgSpeedI = GetValue(66);
    nAvgSpeedF = GetValue(69);
    nETime = GetValue(72)+800;
    nITime = GetValue(75)+850;
    nOffRoadDist1=GetValue(78);
    nOffRoadDist2=GetValue(81);
    nOffRoadTime =
        GetValue(84)*1000+GetValue(87);
    nDrawCL = GetValue(90);

    _lclose(hFileHandle);
}

    // create child window
    lstrcpy((LPSTR) szCaption,
             (LPSTR)szFileName);
    lstrcat ((LPSTR) szCaption,
             " : [Result] ");
    mdicreate.szClass = szRstClass ;
    mdicreate.szTitle = szCaption ;
    mdicreate.hOwner = hInst ;
    mdicreate.x = CW_USEDEFAULT ;
    mdicreate.y = CW_USEDEFAULT ;
mdicreate.cx = CW_USEDEFAULT;
mdicreate.cy = CW_USEDEFAULT;
mdicreate.style = 0;
mdicreate.lpParam = NULL;

hWndChild = (HWND) SendMessage(hWndClient, WM_MDICREATE, 0,
    (long)(LPMDICREATEDSTRUCT)
    &mdicreate);
}
return 0;

// Exit the program
// Messages for arranging windows
// Attempt to close all children
// pass messages to active windows

hWndChild = LOWORD(SendMessage(hWndClient, WM_MDIGETACTIVE, 0, 0L));
if (IsWindow(hWndChild))
    SendMessage(hWndChild, WM_COMMAND, wParam, lParam);
break;  // and then to DefFrameProc
}
broadcast;

case WM_QUERYENDSESSION:
case WM_CLOSE:  // Attempt to close all child windows
    SendMessage ( hWnd, WM_COMMAND, IDM_CLOSEALL, 0L ) ;
    if ( NULL != GetWindow ( hWndClient, GW_CHILD ) )
        return 0;
broadcast;  // ie, call DefFrameProc;

case WM_DESTROY :
    PostQuitMessage ( 0 ) ;
    return 0 ;
}

// Pass unprocessed messages to DefFrameProc

return DefFrameProc ( hWnd, hWndClient, message, wParam, lParam ) ;

// the function of creating copyright window
//----------------------------------------------------
//  CRWndProc()
//----------------------------------------------------
long FAR PASCAL CRWndProc ( hWnd, message, wParam, lParam )
HWND hWnd;
UINT message;
UINT wParam;
LONG lParam;
{
    HDC hCRDC, hCRMemDC;
    HBITMAP hCRBitmap;
    switch ( message )
    {
    case WM_CREATE:
        sWidth = GetSystemMetrics ( SM_CXSCREEN ) ;
        sHeight = GetSystemMetrics ( SM_CYSCREEN ) ;

        // show the window only 6 sec.
        SetTimer( hWnd, 21, 6000, NULL );
        break;
    case WM_PAINT:
        hCRDC = GetDC ( hWnd );
        hCRBitmap = LoadBitmap ( hInst, "CRBITMAP" );
        hCRMemDC = CreateCompatibleDC ( hCRDC );
        SelectObject ( hCRMemDC, hCRBitmap );
BitBlt(hCRDC,(sWidth-500)/2,(sHeight-380)/2,  
500,380,hCRMemDC, 0, 0, SRCCOPY );
DeleteDC ( hCRMemDC );
DeleteObject ( hCRBitmap );
ReleaseDC ( hWnd, hCRDC );
ValidateRect( hWnd, NULL );
break;

case WM_TIMER:
    switch ( wParam )
    {
    case 21:
        KillTimer( hWnd, 21);
        SendMessage (GetParent(hWnd),  
        WM_MDIDESTROY, hWnd, OL) ;
        break;
    }
    break;

case WM_DESTROY:
    return 0 ;

return DefMDIChildProc (hWnd, message, wParam, lParam) ;

// subfunction to close all child windows
//========================================================================
//  CloseEnumProc()
//========================================================================
// Look for all child windows and close them

BOOL FAR PASCAL CloseEnumProc (HWND hWnd, LONG lParam)
{
    if (GetWindow (hWnd, GW_OWNER)) // check for icon title
        return 1 ;
    SendMessage(GetParent(hWnd), WM_MDIRESTORE, hWnd, 0L) ;
    if (!SendMessage (hWnd, WM_QUERYENDSESSION, 0, 0L))
        return 1 ;
    SendMessage (GetParent(hWnd),WM_MDIDESTROY, hWnd, 0L) ;
    return 1 ;
}

//========================================================================
// AboutProc()
// To present a About dialog box

BOOL FAR PASCAL AboutProc ( hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    switch (message)
    {
    case WM_INITDIALOG:
        return (TRUE);

    case WM_COMMAND:
        if (wParam == IDOK)
        {
            EndDialog(hDlg, TRUE);
            return (TRUE);
        }
        break;
    }
    return (FALSE);
}

//============================================================================================================
// TextWndProc()
//============================================================================================================
long FAR PASCAL TextWndProc (HWND hWnd, UINT message, UINT wParam, LONG lParam)
{
    static int nScrollValue=0;
    static int nOldScrollValue=0;
    HPEN hOldPen, hCurrentPen;
    HDC hdc;
    // Local heap to store data
    LOCALHANDLE hDataBuffer;
    NPDATABASE hnpDatabaseBuffer;
    PAINTSTRUCT ps;
    RECT rect;
    static int nP, nU=0;

    // variables listed below are used for printing job

    TEXTMETRIC tm;
    DOCINFO di;
    int nLinePerPage, nCurrentLine, nPageSize;
    int nXChar, nYChar;
    int nErrorCode;
    char szErrorInfo[]="";
    FARPROC lpAbortProc, lpPrintDlgProc;

    switch (message)
case WM_COMMAND:
    if( wParam == IDM_PRINT )
    {
        // Get Print DC

        hPrintDC = GetPrinterDC();
        lpAbortProc = MakeProcInstance((FARPROC)AbortProc, hInst);
        lpPrintDlgProc = MakeProcInstance((FARPROC)PrintDlgProc, hInst);

        if( hPrintDC != NULL )
        {
            bUserCancels = FALSE;
            EnableWindow(hWnd, FALSE);

            // Set "Abort" dialog box

            hPrintDlg = CreateDialog(hInst, "AbortDlg",
                                       hWnd, lpPrintDlgProc);
            SetAbortProc(hPrintDC, lpAbortProc);

            di.cbSize = sizeof(DOCINFO);
            di.lpszDocName="Text Data";
            di.lpszOutput=NULL;

            GetTextMetrics(hPrintDC, &tm);
            nYChar = tm.tmHeight+tm.tmExternalLeading;
            nPageSize=GetDeviceCaps(hPrintDC, VERTRES);
            nLinePerPage=nPageSize/nYChar-1;

            nErrorCode=StartDoc(hPrintDC, &di);

            // if printing error is occurred, send out the
            // error message

            if( nErrorCode < 0 )
            {
                wsprintf(szErrorInfo, "Printing Error! Error
                           No. = %i", nErrorCode);
                BWCCMessageBox(hWnd, szErrorInfo," Print
                           Error", MB_ICONSTOP | MB_OK);
                EnableWindow(hWnd, TRUE);
                DeleteDC(hPrintDC);
            }
            else
            {
                // read data from the local heap, then lock it again
            }
hDataBuffer = GetWindowWord (hWnd, 0);
npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
nRunTimes= npDataBuffer->nRunTimes;
nSubIndex = npDataBuffer->nSubIndex;
nSEG = npDataBuffer->nSEG;
for ( nIndex=0; nIndex<600+1; nIndex++ )
{
    nIndexTmp = nSubIndex+nIndex;
    nLeftRoad[nIndexTmp] = npDataBuffer->
        nLR[nIndex];
    nCarPos[nIndexTmp] = npDataBuffer->nCP[nIndex];
    nCarSpeed[nIndexTmp] = npDataBuffer->
        nCS[nIndex];
}

nP = 0;
StartPage ( hPrintDC );

// write data on the printer device
for ( nIndex=0; nIndex<600+1; nIndex++ )
{
    nIndexTmp = nSubIndex+nIndex;
    if ( nP == 0 )
    {
        wsprintf( szTextOutput, "%4.4i
            %3.3i.%2.2i %3.3i.%2.2i %3.3i",
            nIndexTmp, nLeftRoad[nIndexTmp]/4,
            (nLeftRoad[nIndexTmp]%4)*25,
            nCarPos[nIndexTmp]/4,
            (nCarPos[nIndexTmp]%4)*25,
            nCarSpeed[nIndexTmp] );
        TextOut( hPrintDC, 40,
            nCurrentLine*nYChar,
            szTextOutput, lstrlen(szTextOutput ) );
    }
    else
    {
        wsprintf( szTextOutput, "%
            %3.3i.%2.2i %3.3i.%2.2i %3.3i",
            nCurrentLine*nYChar,
            szTextOutput, lstrlen(szTextOutput ) );
    }
}
nLeftRoad[nIndexTmp]/4,
(nLeftRoad[nIndexTmp]%4)*25,
CarPos[nIndexTmp]/4,
(nCarPos[nIndexTmp]%4)*25,
CarSpeed[nIndexTmp]);
TextOut( hPrintDC, 40, nCurrentLine*nYChar,
szTextOutput, lstrlen( szTextOutput));
}
nP++;

if ( nP == 5 )
{
  nCurrentLine++;
  nP=0;
}

if( ++nCurrentLine > nLinePerPage )
{
  nCurrentLine = 1;
  nErrorCode = EndPage( hPrintDC );

  if( nErrorCode < 0 )
  {
    w sprintf( szErrorInfo, "Printing Error! Error
      No. = %i ", nErrorCode );
    BWCCMessageBox( hWnd, szErrorInfo,
      Print Error",
      MB_ICONSTOP | MB_OK );
    FreeProcInstance ( lpPrintDlgProc );
    FreeProcInstance ( lpAbortProc );
    EnableWindow( hWnd, TRUE );
    DeleteDC ( hPrintDC );
    return FALSE;
  }
  else
    StartPage( hPrintDC );
}

// if printing error is occurred,
// send out the error message

nErrorCode=EndDoc(hPrintDC);
if( nErrorCode < 0 )
{
  w sprintf( szErrorInfo, "Printing Error! Error
    No. = %i ", nErrorCode );
  BWCCMessageBox( hWnd, szErrorInfo," Print
    Error",MB_ICONSTOP | MB_OK );
EnableWindow( hWnd, TRUE );

if( !bUserCancels )
{
    EnableWindow( hWnd, TRUE );
    DestroyWindow( hPrintDlg);
} }
else
    BWCCMessageBox( hWnd,
        "Can not find assigned printer!",
        " Print Error",
        MB_ICONSTOP | MB_OK );

FreeProcInstance (lpPrintDlgProc) ;
FreeProcInstance (lpAbortProc) ;
DeleteDC(hPrintDC);
}

break;

case WM_CREATE:

    lstrcpy((LPSTR)szFileExtention, (LPSTR)szPathTmp);
    if((int)szPathTmp[3]!=0)
        Istrcat((LPSTR)szFileExtention, (LPSTR)"\\ ");
    lstrcat((LPSTR)szFileExtention, (LPSTR)szFileName );
    hFileHandle=_lopen((LPSTR)szFileExtention,0);
    if( hFileHandle != -1 )
        {
            // read data from .data file
            for ( nIndex=0; nIndex<nRunTimes+1; nIndex++ )
            {
                _llseek( hFileHandle, 10*nIndex+12, 0 );
                _lread( hFileHandle, (LPSTR)sz100, 1 );
                _llseek( hFileHandle, 10*nIndex+13, 0 );
                _lread( hFileHandle, (LPSTR)sz10, 1 );
                _llseek( hFileHandle, 10*nIndex+14, 0 );
                _lread( hFileHandle, (LPSTR)sz1, 1 );
                n100=(int)(*sz100)-48;
                n10=(int)(*sz10)-48;
                n1=(int)(*sz1)-48;
                nLeftRoad[nIndex] = n100*100+n10*10+n1;
                _llseek( hFileHandle, 10*nIndex+15, 0 );
                _lread( hFileHandle, (LPSTR)sz100, 1 );
                _llseek( hFileHandle, 10*nIndex+16, 0 );
                _lread( hFileHandle, (LPSTR)sz10, 1 );
                _llseek( hFileHandle, 10*nIndex+17, 0 );
Allocate memory for window private data
switch (nSEG)
{
  case IDD_1SEG:
    nSubIndex = 0;
    break;
  case IDD_2SEG:
    nSubIndex = 600;
    break;
  case IDD_3SEG:
    nSubIndex = 1200;
    break;
  case IDD_4SEG:
    nSubIndex = 1800;
    break;
  case IDD_5SEG:
    nSubIndex = 2400;
    break;
}

// create a local heap to store data
hDataBuffer = LocalAlloc (LMEM_MOVEABLE | LMEM_ZEROINIT,
sizeof (DATABUFFER)) ;

npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer) ;
for( nIndex=0; nIndex<600+1; nIndex++ )
{
    nIndexTmp = nIndex+nSubIndex;
    npDataBuffer->nLR[nIndex] = nLeftRoad[nIndexTmp];
    npDataBuffer->nCP[nIndex] = nCarPos[nIndexTmp];
    npDataBuffer->nCS[nIndex] = nCarSpeed[nIndexTmp];
}

npDataBuffer->nScrollValue = nScrollValue;
npDataBuffer->nRunTimes=nRunTimes;
npDataBuffer->nETime = nETime;
npDataBuffer->nSubIndex = nSubIndex;
npDataBuffer->nSEG = nSEG;
LocalUnlock (hDataBuffer) ;
SetWindowWord (hWnd, 0, hDataBuffer) ;

SetScrollRange (hWnd, SB_VERT, 0, 620, TRUE);
SetScrollPos (hWnd, SB_VERT,
0, /*nOldScrollValue, */ TRUE);

// Save some window handles

hWndClient = GetParent (hWnd) ;
hWndFrame = GetParent (hWndClient) ;
return 0 ;

case WM_VSCROLL:

    // set up everything for vertical scrollbar
    switch(wParam)
    {
    case SB_TOP:
        nScrollValue=0;
        break;
    case SB_BOTTOM:
        nScrollValue=620;
        break;
    case SB_PAGEUP:
        hDataBuffer = GetWindowWord (hWnd, 0) ;
        npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer) ;
        nOldScrollValue = npDataBuffer->nScrollValue;
        LocalUnlock (hDataBuffer) ;
        nOldScrollValue = nScrollValue;
nScrollValue = nOldScrollValue - 25;
break;

case SB_PAGEDOWN:
    hDataBuffer = GetWindowWord ( hWnd, 0 );
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock ( hDataBuffer );
    nOldScrollValue = nScrollValue;
    nScrollValue = nOldScrollValue + 25;
    break;

case SB_LINEUP:
    hDataBuffer = GetWindowWord ( hWnd, 0 );
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock ( hDataBuffer );
    nOldScrollValue = nScrollValue;
    nScrollValue = nOldScrollValue - 1;
    break;

case SB_LINEDOWN:
    hDataBuffer = GetWindowWord ( hWnd, 0 );
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock ( hDataBuffer );
    nOldScrollValue = nScrollValue;
    nScrollValue = nOldScrollValue + 1;
    break;

case SB_THUMBPOSITION:
    hDataBuffer = GetWindowWord ( hWnd, 0 );
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock ( hDataBuffer );
    nOldScrollValue = nScrollValue;
    nScrollValue = LOWORD(lParam);
    break;

}  
if( nScrollValue <= 0 )
    nScrollValue = 0;
if( nScrollValue >= 620 )
    nScrollValue = 620;
if( nScrollValue != nOldScrollValue )
    {  


SetScrollPos(hWnd, SB_VERT, nScrollValue, TRUE);

hDataBuffer = GetWindowWord(hWnd, 0);
npDataBuffer = (NPDATABUFFER) LocalLock(hDataBuffer);
npDataBuffer->nScrollValue = nScrollValue;
LocalUnlock(hDataBuffer);

return 0;

case WM_PAINT:

hdc = BeginPaint(hWnd, &ps);

// read data from local heap

hDataBuffer = GetWindowWord(hWnd, 0);
npDataBuffer = (NPDATABUFFER) LocalLock(hDataBuffer);
nRunTimes = npDataBuffer->nRunTimes;

nSubIndex = npDataBuffer->nSubIndex;
nSEG = npDataBuffer->nSEG;
for(nIndex = 0; nIndex < 600 + 1; nIndex++)
{
  nIndexTmp = nSubIndex + nIndex;
  nLeftRoad[nIndexTmp] = npDataBuffer->nLR[nIndex];
  nCarPos[nIndexTmp] = npDataBuffer->nCP[nIndex];
  nCarSpeed[nIndexTmp] = npDataBuffer->nCS[nIndex];
}

nScrollValue = npDataBuffer->nScrollValue;
LocalUnlock(hDataBuffer);

// paint the screen

GetClientRect(hWnd, &rect);

wsprintf(szTextOutput, "Time Car Car
Time Unit: 0.1 sec."); //,

TextOut(hdc, 40, 5, szTextOutput, lstrlen(szTextOutput));

wsprintf(szTextOutput, "Pos. Speed Road
Speed Unit: mph.");

TextOut(hdc, 40, 20, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "Left Road: lateral
distance between");
TextOut(hdc, 324, 60, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "the origin and left
road edge");
TextOut(hdc, 394, 75, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "Car Pos.: lateral
distance between");
TextOut(hdc, 324, 90, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "the origin and center
car position");
TextOut(hdc, 394, 105, szTextOutput, lstrlen(szTextOutput));

hCurrentPen = CreatePen(PS_SOLID, 3,
RGB(0, 0, 255));
hOldPen = SelectObject(hdc, hCurrentPen);
MoveTo(hdc, 30, 40);
LineTo(hdc, 300, 40);
SelectObject(hdc, hOldPen);
DeleteObject(hCurrentPen);

nU=0;
nP=nScrollValue-(nScrollValue/5)*5;

for(nIndex=nScrollValue; nIndex<600+1; nIndex++)
{
    nIndexTmp=nSubIndex+nIndex;
    if( nP==0 )
    {
        wsprintf(szTextOutput, "%4.4i %3.3i.%2.2i %3.3i.%2.2i %3.3i",
            nIndexTmp, nLeftRoad[nIndexTmp]/4,
            (nLeftRoad[nIndexTmp]%4)*25,
            nCarPos[nIndexTmp]/4,
            (nCarPos[nIndexTmp]%4)*25,
            nCarSpeed[nIndexTmp]);
        TextOut(hdc, 40, (nIndex-nScrollValue+nU)*15+45,
            szTextOutput, lstrlen(szTextOutput));
    }
    else
    {
        wsprintf(szTextOutput, "
            %3.3i.%2.2i %3.3i.%2.2i %3.3i",
            nIndexTmp, nLeftRoad[nIndexTmp]/4,
            (nLeftRoad[nIndexTmp]%4)*25,
            nCarPos[nIndexTmp]/4,
            (nCarPos[nIndexTmp]%4)*25,
            nCarSpeed[nIndexTmp]);
    }
}
nLeftRoad[nIndexTmp]/4,  
(nLeftRoad[nIndexTmp]%4)*25,  
nCarPos[nIndexTmp]/4,  
(nCarPos[nIndexTmp]%4)*25,  
nCarSpeed[nIndexTmp]);
TextOut(hdc, 40, (nIndex-nScrollValue+nU)*15+45,  
szTextOutput, lstrlen(szTextOutput));
}
nP++;  
if (nP == 5)  
{
    nU++;  
nP=0;  
}
EndPaint(hWnd, &ps);
ValidateRect(hWnd, NULL);
return 0;

case WM_QUERYENDSESSION:
    case WM_CLOSE:
        break;  // ie, call DefMDIChildProc
    case WM_DESTROY:
        hDataBuffer = GetWindowWord(hWnd, 0);
        LocalFree(hDataBuffer);
        return 0;
    }
    // Pass unprocessed message to DefMDIChildProc
    return DefMDIChildProc(hWnd, message, wParam, lParam);
}
// the function about graphic display window
//==================================================
// GraphWndProc()
//==================================================
long FAR PASCAL GraphWndProc(HWND hWnd, UINT message, UINT wParam, LONG lParam)
{
    HPEN hOldPen, hCurrentPen;
    HBRUSH hCurrentBrush, hOldBrush;

    // For both side scroll bars
    static int nScrollValue=0;
    // for vertical scroll bar
    static int nOldScrollValue=0;
    static int nhScrollValue=60;
    // for horizontal scroll bar
    static int nOldhScrollValue=30;
HDC hdc, hMemdc;
LOCALHANDLE hDataBuffer; // local heap
NPDATABUFFER npDataBuffer;
PAINTSTRUCT ps;
RECT rect;

// variables for print out
HBITMAP hPrintBitmap;
DOCINFO di;
int nErrorCode;
int nYPageSize, nXPageSize, nXTime, nYTime, nSelected;
char szErrorInfo[] = "";
FARPROC lpAbortProc, lpPrintDlgProc;

switch (message)
{
    case WM_COMMAND:
    {
        if( wParam == IDM_PRINT )
        {
            // get printer device control
            hPrintDC = Get_Printer_DC();
            lpAbortProc = MakeProcInstance ((FARPROC) AbortProc, hInst);
            lpPrintDlgProc = MakeProcInstance ((FARPROC) PrintDlgProc, hInst);

            if( hPrintDC != NULL )
            {
                bUserCancels = FALSE;
                EnableWindow( hWnd, FALSE );

                hPrintDlg = CreateDialog (hInst, "AbortDlg",
                                         hWnd, lpPrintDlgProc);
                SetAbortProc( hPrintDC, lpAbortProc );

                // decide the print out size
                nYPageSize=GetDeviceCaps( hPrintDC, VERTRES );
                nXPageSize=GetDeviceCaps( hPrintDC, HORZRES );
                nXTime = nXPageSize/700;
                nYTime = nYPageSize/800;
                if( nXTime > nYTime )
                    nSelected = nYTime;
                else
                    nSelected = nXTime;
// create a painting area on the memory device control

hdc = GetDC( hWnd );
hMemdc = CreateCompatibleDC( hdc );
hPrintBitmap = CreateCompatibleBitmap ( hdc, 800, 700 );
hOldBitmap=SelectObject( hMemdc, hPrintBitmap );
PatBlt( hMemdc, 0, 0, 800, 700,
        RGB( 255, 255, 255 ));
hDataBuffer = GetWindowWord (hWnd, 0) ;
npDataBuffer = (NPDATABUFFER) LocalLock
             (hDataBuffer) ;
nRunTimes=npDataBuffer->nRunTimes;
nSubIndex=npDataBuffer->nSubIndex;

// read data from the local heap

for( nIndex=0; nIndex<600+1; nIndex++ )
{
    nIndexTmp=nSubIndex+nIndex;
    nLeftRoad[nIndexTmp]=npDataBuffer->nLR[nIndex];
    nCarPos[nIndexTmp]=npDataBuffer->nCP[nIndex];
    nCarSpeed[nIndexTmp]=npDataBuffer->nCS[nIndex];
}
nScrollValue=npDataBuffer->nScrollValue;
nhScrollValue=npDataBuffer->nhScrollValue;
nRWft=npDataBuffer->nRWft;
nOW=npDataBuffer->nOW;
LocalUnlock (hDataBuffer) ;

hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen = SelectObject( hMemdc, hCurrentPen );

// paint road on the memory device
for( nIndex=600; nIndex>=0; nIndex-- )
{
    nIndexTmp=nSubIndex+nIndex;
    MoveTo( hMemdc, nLeftRoad[nIndexTmp], 40+600-
            nIndex );
    LineTo( hMemdc, nLeftRoad[nIndexTmp]+nRWft*4,
            40+600-nIndex );

    if( (nIndexTmp - (nIndexTmp/50)*50) == 0 )
    {
        wsprintf ( szTextOutput, "%4i", nIndexTmp);
        TextOut( hMemdc, 0, 40+600-nIndex ,
                 szTextOutput,
lstrlen( szTextOutput ));
}
}
SelectObject( hMemdc, hOldPen );
DeleteObject( hCurrentPen );

// paint the object's route on the memory device
hCurrentPen = CreatePen( PS_SOLID, 1,
    RGB(255,255,0) );
hOldPen=SelectObject( hMemdc, hCurrentPen );
for( nIndex=600; nIndex>=0; nIndex-- )
{
    nIndexTmp=nSubIndex+nIndex;
    MoveTo( hMemdc, nCarPos[nIndexTmp], 40+600-
    nIndex );
    LineTo( hMemdc, nCarPos[nIndexTmp]+nOW, 40+600-
    nIndex );
}
SelectObject( hMemdc, hOldPen );
DeleteObject( hCurrentPen );

// paint object speed on the memory device
hCurrentPen = CreatePen( PS_SOLID, 1,
    RGB(0,255,255) );
hOldPen=SelectObject( hMemdc, hCurrentPen );
for( nIndex=600; nIndex>=0; nIndex-- )
{
    nIndexTmp=nSubIndex+nIndex;
    MoveTo( hMemdc, 520, 40+600-nIndex );
    LineTo( hMemdc, 520+nCarSpeed[nIndex], 40+600-
    nIndex );
}
SelectObject( hMemdc, hOldPen );
DeleteObject( hCurrentPen );

hCurrentPen = CreatePen( PS_SOLID, 1,
    RGB(0,0,255) );
hOldPen=SelectObject( hMemdc, hCurrentPen );

// write scale on the memory device
MoveTo( hMemdc, 520+30, 40 );
LineTo( hMemdc, 520+30, 40+600-nIndex );
MoveTo( hMemdc, 520+60, 40 );
LineTo( hMemdc, 520+60, 40+600-nIndex );
MoveTo( hMemdc, 520+90, 40 );
LineTo( hMemdc, 520+90, 40+600-nIndex );
MoveTo( hMemdc, 520+120, 40 );
LineTo( hMemdc, 520+120, 40+600-nIndex );
MoveTo( hMemdc, 520+150, 40 );
LineTo( hMemdc, 520+150, 40+600-nIndex );
SelectObject( hMemdc, hOldPen );
DeleteObject( hCurrentPen );

hCurrentBrush = CreateSolidBrush( RGB(0,0,0));
hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen=SelectObject( hMemdc, hCurrentPen );
hOldBrush=SelectObject( hMemdc, hCurrentBrush );
Rectangle (hMemdc,120+50-10, 5, 120+100-10, 20);
SelectObject( hMemdc, hOldBrush );
DeleteObject( hCurrentBrush );
SelectObject( hMemdc, hOldPen );
DeleteObject( hCurrentPen );

hCurrentBrush = CreateSolidBrush( RGB(255,255,0));
hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen=SelectObject( hMemdc, hCurrentPen );
hOldBrush=SelectObject( hMemdc, hCurrentBrush );
Rectangle (hMemdc,320+80-10, 5, 320+130-10, 20);
SelectObject( hMemdc, hOldBrush );
DeleteObject( hCurrentBrush );
SelectObject( hMemdc, hOldPen );
DeleteObject( hCurrentPen );

// write titles on the memory device

wsprintf( szTextOutput, "Time");
TextOut( hMemdc, 0, 5, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "Car Route");
TextOut( hMemdc, 320+0-10, 5, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "Road");
TextOut( hMemdc, 120+0-10, 5, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "Car Speed");
TextOut( hMemdc, 520+0-10, 5, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%1.1i", 0 );
TextOut( hMemdc, 520+0-10, 23, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%2.2i", 30 );
TextOut( hMemdc, 520+30-10, 23, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%2.2i", 60 );
TextOut( hMemdc, 520+60-10, 23, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "$2.2i", 90 );
TextOut( hMemdc, 520+90-10, 23, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "$3.3i", 120 );
TextOut( hMemdc, 520+120-10, 23, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "$3.3i", 150 );
TextOut( hMemdc, 520+150-10, 23, szTextOutput, lstrlen( szTextOutput ));

// set document information
di.cbSize = sizeof(DOCINFO);
di.lpszDocName="Graph. Data";
di.lpszOutput=NULL;

nErrorCode=StartDoc( hPrintDC, &di );

// check error message
if( nErrorCode > 0 )
{
    StretchBlt( hPrintDC, 0, 0, nSelected*800, nSelected*700, hMemdc, 0, 0, 800, 700, SRCCOPY );
nErrorCode=Escape( hPrintDC, NEWFRAME, 0, NULL );
    if( nErrorCode > 0 )
    {
        Escape( hPrintDC, ENDDOC, strlen("Result") , "Result", NULL );
    }
else
    {
        wsprintf( szErrorInfo, "Printing Error! Error No. = %i ", nErrorCode );
        BWCCMessageBox( hWnd, szErrorInfo," Print Error", MB_ICONSTOP | MB_OK );
    }
else
    {
        wsprintf( szErrorInfo, "Printing Error! Error No. = %i ", nErrorCode );
        BWCCMessageBox( hWnd, szErrorInfo," Print Error", MB_ICONSTOP | MB_OK );
    }
}
if( !bUserCancels )
{
    EnableWindow( hWnd, TRUE );
    DestroyWindow( hPrintDlg);
}

SelectObject( hMemdc, hOldBitmap );
DeleteObject( hPrintBitmap);
DeleteDC( hMemdc );
ReleaseDC( hWnd, hdc );
}
else

    BWCCMessageBox( hWnd,
        "Can not find assigned printer!",
        " Print Error",
        MB_ICONSTOP | MB_OK);

FreeProcInstance ( lpPrintDlgProc );
FreeProcInstance ( lpAbortProc );
DeleteDC(hPrintDC);

break;

case WM_CREATE:

    // Allocate memory for window private data

    lstrcpy((LPSTR)szFileExtention, (LPSTR)szPathTmp );
    if((int)szPathTmp[3]! =0)
        lstrcat((LPSTR)szFileExtention, (LPSTR)"\"");
    lstrcat((LPSTR)szFileExtention, (LPSTR)szFileName );
    hFileHandle=_lopen((LPSTR)szFileExtention,0);
    if ( hFileHandle != -1 )
    {
        // read data from .data file
        for ( nIndex=0; nIndex<nRunTimes+1; nIndex++ )
        {
            _lseek( hFileHandle, 10*nIndex+12, 0 );
            _lread( hFileHandle, (LPSTR)sz100, 1 );
            _lseek( hFileHandle, 10*nIndex+13, 0 );
            _lread( hFileHandle, (LPSTR)sz10, 1 );
            _lseek( hFileHandle, 10*nIndex+14, 0 );
            _lread( hFileHandle, (LPSTR)sz1, 1 );
            n100=(int)(*sz100)-48;
            n10=(int)(*sz10)-48;
            n1=(int)(*sz1)-48;
            nLeftRoad[nIndex] = n100*100+n10*10+n1;
switch(nSEG)
{
    case IDD_1SEG:
        nSubIndex = 0;
        break;
    case IDD_2SEG:
        nSubIndex = 600;
        break;
    case IDD_3SEG:
        nSubIndex = 1200;
        break;
    case IDD_4SEG:
        nSubIndex = 1800;
        break;
    case IDD_5SEG:
        nSubIndex = 2400;
        break;
}

SetScrollRange (hWnd, SB_VERT, 0, 620, TRUE);
SetScrollPos (hWnd, SB_VERT, nOldScrollValue, TRUE);
SetScrollRange (hWnd, SB_HORZ, 0, 60, TRUE);
SetScrollPos (hWnd, SB_HORZ, nOldhScrollValue, TRUE);

hDataBuffer = LocalAlloc (LMEM_MOVEABLE | LMEM_ZEROINIT, sizeof (DATABUFFER)) ;

npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);

npDataBuffer->nSubIndex = nSubIndex;
for( nIndex=0; nIndex<600+1; nIndex++)
{
    nIndexTmp=nSubIndex+nIndex;
    npDataBuffer->nLR[nIndex] = nLeftRoad[nIndexTmp];
    npDataBuffer->nCP[nIndex] = nCarPos[nIndexTmp];
    npDataBuffer->nCS[nIndex] = nCarSpeed[nIndexTmp];
}

npDataBuffer->nScrollValue = nScrollValue;
npDataBuffer->nhScrollValue = nhScrollValue;
npDataBuffer->nRWft = nRWft;
npDataBuffer->nOW = nOW;
npDataBuffer->nRunTimes = nRunTimes;
LocalUnlock (hDataBuffer);
SetWindowWord (hWnd, 0, hDataBuffer);

// Save some window handles

hWndClient = GetParent (hWnd);
hWndFrame = GetParent (hWndClient);
return 0;

case WM_VSCROLL:

    // Vertical scroll bar

    switch (wParam)
    {
    case SB_TOP:
        nScrollValue=0;
        break;
case SB_BOTTOM:
    nScrollValue = 620;
    break;

case SB_PAGEUP:
    hDataBuffer = GetWindowWord (hWnd, 0);
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock (hDataBuffer);
    nScrollValue = nOldScrollValue - 10;
    break;

case SB_PAGEDOWN:
    hDataBuffer = GetWindowWord (hWnd, 0);
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock (hDataBuffer);
    nScrollValue = nOldScrollValue + 10;
    break;

case SB_LINEUP:
    hDataBuffer = GetWindowWord (hWnd, 0);
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock (hDataBuffer);
    nScrollValue = nOldScrollValue + 1;
    break;

case SB_LINEDOWN:
    hDataBuffer = GetWindowWord (hWnd, 0);
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock (hDataBuffer);
    nScrollValue = nOldScrollValue + 1;
    break;

case SB_THUMBPOSITION:
    hDataBuffer = GetWindowWord (hWnd, 0);
    npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
    nOldScrollValue = npDataBuffer->nScrollValue;
    LocalUnlock (hDataBuffer);
    nScrollValue = LOWORD(lParam);
    break;
if( nScrollValue <= 0 )
    nScrollValue = 0;
if( nScrollValue >= 620 )
    nScrollValue = 620;
if( nScrollValue != nOldScrollValue )
{
    SetScrollPos(hWnd, SB_VERT, nScrollValue, TRUE);
    hDataBuffer = GetWindowWord (hWnd, 0);
    npDataBuffer = (NPDATABUFFER) LocalLock(hDataBuffer);
    npDataBuffer->nScrollValue = nScrollValue;
    LocalUnlock (hDataBuffer);

    // repaint client area

    InvalidateRect(hWnd, NULL, TRUE);
}
return 0;

case WM_HSCROLL:

    // for horizontal scroll bar

    switch(wParam)
    {
    case SB_TOP:
        nhScrollValue=0;
        break;

    case SB_BOTTOM:
        nhScrollValue=60;
        break;

    case SB_PAGEUP:
        hDataBuffer = GetWindowWord (hWnd, 0);
        npDataBuffer = (NPDATABUFFER) LocalLock(hDataBuffer);
        nOldhScrollValue = npDataBuffer->nhScrollValue;
        LocalUnlock (hDataBuffer);
        nhScrollValue=nOldhScrollValue-10;
        break;

    case SB_PAGEDOWN:
        hDataBuffer = GetWindowWord (hWnd, 0);
        npDataBuffer = (NPDATABUFFER) LocalLock(hDataBuffer);

nOldhScrollValue = npDataBuffer->nhScrollValue;
LocalUnlock (hDataBuffer) ;
nhScrollValue=nOldhScrollValue+10;
break;

case SB_LINEUP:
hDataBuffer = GetWindowWord (hWnd, 0) ;
npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
nOldhScrollValue = npDataBuffer->nhScrollValue;
LocalUnlock (hDataBuffer) ;
nhScrollValue=nOldhScrollValue-1;
break;

case SB_LINEDOWN:
hDataBuffer = GetWindowWord (hWnd, 0) ;
npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
nOldhScrollValue = npDataBuffer->nhScrollValue;
LocalUnlock (hDataBuffer) ;
nhScrollValue=nOldhScrollValue+1;
break;

case SB_THUMBPOSITION:
hDataBuffer = GetWindowWord (hWnd, 0) ;
npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
nOldhScrollValue = npDataBuffer->nhScrollValue;
LocalUnlock (hDataBuffer) ;
nhScrollValue=LOWORD(lParam);
break;
}

if( nhScrollValue <= 0 )
nhScrollValue = 0;
if( nhScrollValue >= 60 )
nhScrollValue = 60;
if( nhScrollValue != nOldhScrollValue )
{
    SetScrollPos(hWnd, SB_HORZ, nhScrollValue, TRUE);
hDataBuffer = GetWindowWord (hWnd, 0) ;
npDataBuffer = (NPDATABUFFER) LocalLock (hDataBuffer);
npDataBuffer->nhScrollValue = nhScrollValue;
LocalUnlock (hDataBuffer) ;
}
InvalidateRect(hWnd, NULL, TRUE);
}
return 0;

case WM_DESTROY:
hDataBuffer = GetWindowWord (hWnd, 0) ;
LocalFree (hDataBuffer) ;
return 0 ;

case WM_PAINT:

    // Paint the window
    hdc = BeginPaint (hWnd, &ps) ;
    // the procedure is similar to the printing
    // procedure

    hDataBuffer = GetWindowWord (hWnd, 0) ;
    npDataBuffer = (NPDATABUFFER) LocalLock
                   (hDataBuffer) ;
    nRunTimes=npDataBuffer->nRunTimes;
    nSubIndex=npDataBuffer->nSubIndex;

    for( nIndex=0; nIndex<600+1; nIndex++ )
    {
        nIndexTmp=nSubIndex+nIndex;
        nLeftRoad[nIndexTmp]=npDataBuffer->nLR[nIndex];
        nCarPos[nIndexTmp]=npDataBuffer->nCP[nIndex];
        nCarSpeed[nIndexTmp]=npDataBuffer->nCS[nIndex];
    }
    nScrollValue=npDataBuffer->nScrollValue;
    nhScrollValue=npDataBuffer->nhScrollValue;
    nRWft=npDataBuffer->nRWft;
    nOW=npDataBuffer->nOW;

    LocalUnlock (hDataBuffer) ;

    hCurrentPen = GetStockObject( BLACK_PEN ) ;
hOldPen = SelectObject( hdc, hCurrentPen ) ;

    GetClientRect (hWnd, &rect) ;
    for(nIndex=600-nScrollValue;nIndex>=0;nIndex-- )
    {
        nIndexTmp=nSubIndex+nIndex;
        MoveTo( hdc, nLeftRoad[nIndexTmp]-nhScrollValue,
                00-nScrollValue-nIndex ) ;
        LineTo( hdc, nLeftRoad[nIndexTmp]+nRWft*4-
                nhScrollValue,
                40+600-nScrollValue-nIndex ) ;
    }

}
if( (nIndexTmp - (nIndexTmp/50)*50) == 0 )
{
    wsprintf( szTextOutput, "%4i", nIndexTmp);
    TextOut( hdc, 0-nhScrollValue, 40+600-nScrollValue-nIndex, szTextOutput, lstrlen(szTextOutput));
}

SelectObject( hdc, hOldPen );
DeleteObject( hCurrentPen );

hCurrentPen = CreatePen( PS_SOLID, 1, RGB(255,255,0) );
hOldPen=SelectObject( hdc, hCurrentPen );
for(nIndex=600-nScrollValue;nIndex>=0;nIndex-- )
{
    nIndexTmp=nSubIndex+nIndex;
    MoveTo( hdc, nCarPos[nIndexTmp]-nhScrollValue, 40+600-nScrollValue-nIndex );
    LineTo( hdc, nCarPos[nIndexTmp]+nOW-nhScrollValue, 40+600-nScrollValue-nIndex );
}

SelectObject( hdc, hOldPen );
DeleteObject( hCurrentPen );

hCurrentPen = CreatePen( PS_SOLID, 1, RGB(0,255,255) );
hOldPen=SelectObject( hdc, hCurrentPen );
for(nIndex=600-nScrollValue;nIndex>=0;nIndex-- )
{
    nIndexTmp=nSubIndex+nIndex;
    MoveTo( hdc, 520-nhScrollValue, 40+600-nScrollValue-nIndex );
    LineTo( hdc, 520+nCarSpeed[nIndexTmp]-nScrollValue, 40+600-nScrollValue-nIndex );
}

SelectObject( hdc, hOldPen );
DeleteObject( hCurrentPen );

hCurrentPen = CreatePen( PS_SOLID, 1, RGB(0,0,255) );
hOldPen=SelectObject( hdc, hCurrentPen );
MoveTo( hdc, 520+30-nhScrollValue, 40 );
LineTo( hdc, 520+30-nhScrollValue, 40+600-nScrollValue-nIndex );
MoveTo( hdc, 520+60-nhScrollValue, 40 );
LineTo( hdc, 520+60-nhScrollValue, 40+600-nScrollValue-nIndex );
MoveTo( hdc, 520+90-nhScrollValue, 40 );
LineTo( hdc, 520+90-nhScrollValue,
          40+600-nScrollValue-nIndex );
MoveTo( hdc, 520+120-nhScrollValue, 40 );
LineTo( hdc, 520+120-nhScrollValue,
          40+600-nScrollValue-nIndex );
MoveTo( hdc, 520+150-nhScrollValue, 40 );
LineTo( hdc, 520+150-nhScrollValue,
          40+600-nScrollValue-nIndex );
SelectObject( hdc, hOldPen );
DeleteObject( hCurrentPen );

hCurrentBrush = CreateSolidBrush( RGB(0,0,0));
hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen=SelectObject( hdc, hCurrentPen );
hOldBrush=SelectObject( hdc, hCurrentBrush );
Rectangle ( hdc, 120+50-10-nhScrollValue, 5,
          120+100-10-nhScrollValue, 20);
SelectObject( hdc, hOldBrush );
DeleteObject( hCurrentBrush );
SelectObject( hdc, hOldPen );
DeleteObject( hCurrentPen );

hCurrentBrush = CreateSolidBrush( RGB(255,255,0));
hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen=SelectObject( hdc, hCurrentPen );
hOldBrush=SelectObject( hdc, hCurrentBrush );
Rectangle ( hdc, 320+80-10-nhScrollValue, 5,
          320+130-10-nhScrollValue, 20);
SelectObject( hdc, hOldBrush );
DeleteObject( hCurrentBrush );
SelectObject( hdc, hOldPen );
DeleteObject( hCurrentPen );

wsprintf( szTextOutput, "Time");
TextOut( hdc, 0-nhScrollValue, 5, szTextOutput,
        lstrlen( szTextOutput ));

wsprintf( szTextOutput, "Car Route");
TextOut( hdc, 320+0-10-nhScrollValue, 5,
        szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "Road");
TextOut( hdc, 120+0-10-nhScrollValue, 5,
        szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "Car Speed");
TextOut( hdc, 520+0-10-nhScrollValue, 5,
        szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%1.1i", 0 );
TextOut(hdc, 520 + 0 - 10 - nhScrollValue, 23, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "%2.2i", 30);
TextOut(hdc, 520 + 30 - 10 - nhScrollValue, 23, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "%2.2i", 60);
TextOut(hdc, 520 + 60 - 10 - nhScrollValue, 23, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "%2.2i", 90);
TextOut(hdc, 520 + 90 - 10 - nhScrollValue, 23, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "%3.3i", 120);
TextOut(hdc, 520 + 120 - 10 - nhScrollValue, 23, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "%3.3i", 150);
TextOut(hdc, 520 + 150 - 10 - nhScrollValue, 23, szTextOutput, lstrlen(szTextOutput));

EndPaint(hWnd, &ps);
ValidateRect(hWnd, NULL);
return 0;

// Pass unprocessed message to DefMDIChildProc
return DefMDIChildProc(hWnd, message, wParam, lParam);

} // the function of the information window

InfoWndProc() {
// variables to write data on the screen
HPEN hOldPen, hCurrentPen;
HDC hdc;
LOCALHANDLE hInfoBuffer;
NPINFOBUFFER npInfoBuffer;
PAINTSTRUCT ps;
RECT rect;

// variables for printing out
TEXTMETRIC tm;
DOCINFO di;
int nLinePerPage, nCurrentLine, nPageSize;
int nXChar, nYChar;
int nErrorCode;
char szErrorInfo[] = "";
FARPROC lpAbortProc, lpPrintDlgProc;

switch (message) {
    case WM_COMMAND:
        if (wParam == IDM_PRINT) {
            // get printer DC
            hPrintDC = GetPrinterDC();
            lpAbortProc = MakeProcInstance((FARPROC)AbortProc, hInst);
            lpPrintDlgProc = MakeProcInstance((FARPROC)PrintDlgProc, hInst);

            // set a "Abort" procedure
            if (hPrintDC != NULL) {
                bUserCancels = FALSE;
                EnableWindow(hWnd, FALSE);

                hPrintDlg = CreateDialog(hInst, "AbortDlg", hWnd, lpPrintDlgProc);
                SetAbortProc(hPrintDC, lpAbortProc);

                di.cbSize = sizeof(DOCINFO);
                di.lpszDocName="Text Data";
                di.lpszOutput=NULL;

                GetTextMetrics(hPrintDC, &tm);
                nYChar = tm.tmHeight+tm.tmExternalLeading;

                nErrorCode=StartDoc(hPrintDC, &di);

                if (nErrorCode < 0) {
                    wsprintf(szErrorInfo, "Printing Error! Error No. = %i", nErrorCode);
                    BWCCMessageBox(hWnd, szErrorInfo, "Print Error", MB_ICONSTOP | MB_OK);

                    EnableWindow(hWnd, TRUE);
                    DeleteDC(hPrintDC);
                } else
                {

                    EnableWindow(hWnd, TRUE);
                    DeleteDC(hPrintDC);
                }
            }
        }
}
Ilread data from the local heap

StartPage( hPrintDC );

hInfoBuffer = GetWindowWord ( hWnd, 0 );
npInfoBuffer = (NPINFOBUFFER) LocalLock (hInfoBuffer);

{ 
 nPreDist = npInfoBuffer->nBuff[0];
nRoadWidthft = npInfoBuffer->nBuff[1];
nSimultSpeed = npInfoBuffer->nBuff[2];
nSpeedVariable = npInfoBuffer->nBuff[3];
nCurves = npInfoBuffer->nBuff[4];
nLags = npInfoBuffer->nBuff[5];
nGains = npInfoBuffer->nBuff[6];
ObjectShape = npInfoBuffer->nBuff[7];
DrawCL = npInfoBuffer->nBuff[8];
ETime = npInfoBuffer->nBuff[9];
ITime = npInfoBuffer->nBuff[10];
CurveDir = npInfoBuffer->nBuff[11];
PreviewSelect = npInfoBuffer->nBuff[12];
PreviewTime = npInfoBuffer->nBuff[13];
for( nIndex=0; nIndex<10; nIndex++ )
{
 szSeed[nIndex]= npInfoBuffer->szSd[nIndex];
}
for( nIndex=0; nIndex<20; nIndex++ )
{
 szSubjectID[nIndex]= npInfoBuffer-> szSID[nIndex];
}
LocalUnlock (hInfoBuffer);

// write data on the printer device
wsprintf( szTextOutput, "Subject ID : %s ", (LPSTR)szSubjectID);
TextOut( hPrintDC, 40, newYChar, szTextOutput, lstrlen(szTextOutput));

if( nPreviewSelect == 1101 )
wsprintf( szTextOutput, "Preview Distance : %i ft.", nPreDist ) ;
else
{
 switch( nPreviewTime )
{
case 1402:
 lstrcpy( (LPSTR)szTextOutput, (LPSTR)"Preview Time : 0.2 sec." );
break;

case 1404:
    lstrcpy((LPSTR)szTextOutput,
            (LPSTR)"Preview Time : 0.4 sec.");
    break;

case 1408:
    lstrcpy((LPSTR)szTextOutput,
            (LPSTR)"Preview Time : 0.8 sec.");
    break;

case 1410:
    lstrcpy((LPSTR)szTextOutput,
            (LPSTR)"Preview Time : 1.0 sec.");
    break;

case 1420:
    lstrcpy((LPSTR)szTextOutput,
            (LPSTR)"Preview Time : 2.0 sec.");
    break;

case 1432:
    lstrcpy((LPSTR)szTextOutput,
            (LPSTR)"Preview Time : 3.2 sec.");
    break;

case 1450:
    lstrcpy((LPSTR)szTextOutput,
            (LPSTR)"Preview Time : 5.0 sec.");
    break;

case 1464:
    lstrcpy((LPSTR)szTextOutput,
            (LPSTR)"Preview Time : 6.4 sec.");
    break;
}

TextOut(hPrintDC, 40, 2*nYChar,
        szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "Road Width : %i ft.", nRoadWidthft);
TextOut(hPrintDC, 40, 3*nYChar,
        szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "Initial Speed : %i mph.", nSimultSpeed);
TextOut(hPrintDC, 40, 4*nYChar,
        szTextOutput, lstrlen(szTextOutput));
wsprintf( szTextOutput, "Initial Seed : %s", (LPSTR)szSeed );
TextOut( hPrintDC, 40, 5*nYChar,
        szTextOutput, lstrlen( szTextOutput ) );

if( nSpeedVariable == 750 )
    lstrcpy ((LPSTR)szTextOutput, "Object Speed: Constant" );
else
    lstrcpy ((LPSTR)szTextOutput, "Object Speed: Variable" );
TextOut( hPrintDC, 40, 6*nYChar,
        szTextOutput, lstrlen( szTextOutput ) );

wsprintf( szTextOutput, "Relative Curve Rate : %i %%", nCurves );
TextOut( hPrintDC, 40, 7*nYChar,
        szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "Time Lag : %i",
        nLags );
TextOut( hPrintDC, 40, 8*nYChar,
        szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "Input Gain : %i",
        nGains );
TextOut( hPrintDC, 40, 9*nYChar,
        szTextOutput, lstrlen( szTextOutput ) );

if( nObjectShape == 907 )
    lstrcpy ((LPSTR)szTextOutput,"Object Shape : Car" );
else
    lstrcpy ((LPSTR)szTextOutput,"Object Shape : Circle" );
TextOut( hPrintDC, 40, 10*nYChar,
        szTextOutput, lstrlen( szTextOutput ) );

if( nDrawCL == 909 )
    lstrcpy ((LPSTR)szTextOutput,(LPSTR)"Center Line : Yes" );
else
    lstrcpy ((LPSTR)szTextOutput,(LPSTR)"Center Line : No" );
TextOut( hPrintDC, 40, 11*nYChar,
        szTextOutput, lstrlen( szTextOutput ) );

switch( nETime )
{
    case 801:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Experiment Time : 1 min." );
break;

case 802:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Experiment Time : 2 min." );
break;

case 803:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Experiment Time : 3 min." );
break;

case 804:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Experiment Time : 4 min." );
break;

case 805:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Experiment Time : 5 min." );
break;
}

TextOut( hPrintDC, 40, 12*nYChar, szTextOutput, lstrlen( szTextOutput) );

switch( nITime )
{

case 851:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time : 5 sec." );
'break;


case 852:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time : 10 sec." );
break;


case 853:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time : 15 sec." );
break;


case 854:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time : 20 sec." );
break;

case 855:
    lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time : 30 sec." );
    break;
}
TextOut( hPrintDC, 40, 13*nYChar, szTextOutput, lstrlen( szTextOutput) );

if( nCurveDir == 870 )
    lstrcpy ((LPSTR)szTextOutput, (LPSTR)"First Curve : Right" );
else
    lstrcpy ((LPSTR)szTextOutput, (LPSTR)"First Curve : Left" );
TextOut( hPrintDC, 40, 14*nYChar, szTextOutput, lstrlen( szTextOutput) );
if( nPreviewSelect == 1101 )
    lstrcpy( (LPSTR)szTextOutput,
             (LPSTR)"Preview Selection : Constant Distance" );
else
    lstrcpy( (LPSTR)szTextOutput,
             (LPSTR)"Preview Selection : Constant Time" );
TextOut( hPrintDC, 40, 15*nYChar,
          szTextOutput, lstrlen( szTextOutput) );

// error code
nErrorCode=EndDoc(hPrintDC);
if( nErrorCode < 0 )
{
    wsprintf( szErrorInfo, "Printing Error! Error No. = %i ", nErrorCode );
    BWCCMessageBox( hWnd, szErrorInfo," Print Error", MB_ICONSTOP | MB_OK );
    EnableWindow( hWnd, TRUE );
}

if( !bUserCancels )
{
    EnableWindow( hWnd, TRUE );
    DestroyWindow( hPrintDlg);
else
    BWCCMessageBox( hWnd, 
        "Can not find assigned printer!", 
        " Print Error", 
        MB_ICONSTOP | MB_OK );

FreeProcInstance (lpPrintDlgProc); 
FreeProcInstance (lpAbortProc); 
DeleteDC(hPrintDC);
}
break;

case WM_CREATE:

    // Allocate memory for window private data

    hInfoBuffer = LocalAlloc (LMEM_MOVEABLE | 
        LMEM_ZEROINIT,sizeof (INFOBUFFER)); 
    npInfoBuffer = (NPINFOBUFFER)
    LocalLock(hInfoBuffer);
{
    npInfoBuffer->nBuff[0] = nPreDist;
    npInfoBuffer->nBuff[1] = nRoadWidthft;
    npInfoBuffer->nBuff[2] = nSimultSpeed;
    npInfoBuffer->nBuff[3] = nSpeedVariable;
    npInfoBuffer->nBuff[4] = nCurves;
    npInfoBuffer->nBuff[5] = nLags;
    npInfoBuffer->nBuff[6] = nGains;
    npInfoBuffer->nBuff[7] = nObjectShape;
    npInfoBuffer->nBuff[8] = nDrawCL;
    npInfoBuffer->nBuff[9] = nETime;
    npInfoBuffer->nBuff[10] = nITime;
    npInfoBuffer->nBuff[12] = nPreviewSelect;
    for( nIndex=0; nIndex<10; nIndex++ )
    {
        npInfoBuffer->szSd[nIndex] = szSeed[nIndex];
    }
    for( nIndex=0; nIndex<20; nIndex++ )
    {
        npInfoBuffer->szSID[nIndex]=szSubjectID[nIndex];
    }
}
LocalUnlock (hInfoBuffer); 
SetWindowWord ( hWnd, 0, hInfoBuffer );
// Save some window handles

dc = GetDC( hWnd );
hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen = SelectObject( hdc, hCurrentPen );
ReleaseDC( hWnd, hdc );

hWndClient = GetParent( hWnd );
hWndFrame = GetParent( hWndClient );
return 0 ;

case WM_DESTROY:

dc = GetDC( hWnd );
SelectObject( hdc, hOldPen );
DeleteObject( hCurrentPen );
ReleaseDC( hWnd, hdc );

hInfoBuffer = GetWindowWord (hWnd, 0 );

// free the local heap

LocalFree (hInfoBuffer);
return 0 ;

case WM_PAINT:

// Paint the window

hc = BeginPaint (hWnd, &ps );

hInfoBuffer = GetWindowWord (hWnd, 0 );
nInfoBuffer = (NPINFOBUFFER) LocalLock (hInfoBuffer);
{
  nPreDist = npInfoBuffer->nBuff[0];
nRoadWidthft = npInfoBuffer->nBuff[1];
nSimultSpeed = npInfoBuffer->nBuff[2];
nSpeed = npInfoBuffer->nBuff[3];
nCurves = npInfoBuffer->nBuff[4];
nLags = npInfoBuffer->nBuff[5];
nGains = npInfoBuffer->nBuff[6];
nObjectShape = npInfoBuffer->nBuff[7];
nDrawCL = npInfoBuffer->nBuff[8];
nETime = npInfoBuffer->nBuff[9];
nITime = npInfoBuffer->nBuff[10];
nCurveDir = npInfoBuffer->nBuff[11];
nPreviewSelect = npInfoBuffer->nBuff[12];
nPreviewTime = npInfoBuffer->nBuff[13];
for( nIndex=0; nIndex<10; nIndex++ )
{
    szSeed[nIndex] = npInfoBuffer->szSd[nIndex];
}
for( nIndex=0; nIndex<20; nIndex++ )
{
    szSubjectID[nIndex] = npInfoBuffer->szSID[nIndex];
}
}
LocalUnlock (hInfoBuffer);

hCurrentPen = GetStockObject( BLACK_PEN );

// the procedure is similar to the procedure of
// writing data on the window
GetClientRect (hWnd, &rect);
SelectObject( hdc, hCurrentPen );

wsprintf( szTextOutput, "Subject ID : %s ",
            (LPSTR)szSubjectID );
TextOut( hdc, 10, 0, szTextOutput, lstrlen( szTextOutput ));

if( nPreviewSelect == 1101 )
    wsprintf( szTextOutput, "Preview Distance : %i
            ft. ", nPreDist );
else
{
    switch( nPreviewTime )
    {
    case 1402:
        lstrcpy( (LPSTR)szTextOutput,
                (LPSTR)"Preview Time : 0.2 sec." );
        break;

    case 1404:
        lstrcpy( (LPSTR)szTextOutput,
                (LPSTR)"Preview Time : 0.4 sec." );
        break;

    case 1408:
        lstrcpy( (LPSTR)szTextOutput,
                (LPSTR)"Preview Time : 0.8 sec." );
        break;

    case 1410:
lstrcpy( (LPSTR)szTextOutput, (LPSTR)"Preview Time : 1.0 sec." );
break;

case 1420:
lstrcpy( (LPSTR)szTextOutput, (LPSTR)"Preview Time : 2.0 sec." );
break;

case 1432:
lstrcpy( (LPSTR)szTextOutput, (LPSTR)"Preview Time : 3.2 sec." );
break;

case 1450:
lstrcpy( (LPSTR)szTextOutput, (LPSTR)"Preview Time : 5.0 sec." );
break;

case 1464:
lstrcpy( (LPSTR)szTextOutput, (LPSTR)"Preview Time : 6.4 sec." );
break;

}

TextOut( hdc, 10, 20, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "Road Width : %i ft.", nRoadWidthft);
TextOut( hdc, 10, 40, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "Initial Speed : %i mph.", nSimultSpeed );
TextOut( hdc, 10, 60, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "Initial Seed : %S", (LPSTR)szSeed );
TextOut( hdc, 10, 80, szTextOutput, lstrlen( szTextOutput ) );

if( nSpeedVariable == 750 )
lstrcpy( (LPSTR)szTextOutput, "Object Speed: Constant" );
else
lstrcpy( (LPSTR)szTextOutput, "Object Speed : Variable" );
TextOut( hdc, 10, 100, szTextOutput, lstrlen( szTextOutput ) );
wsprintf(szTextOutput, "Relative Curve Rate: %i %%", nCurves);
TextOut(hdc, 10, 120, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "Time Lag: %i", nLags);
TextOut(hdc, 10, 140, szTextOutput, lstrlen(szTextOutput));
wsprintf(szTextOutput, "Input Gain: %i", nGains);
TextOut(hdc, 10, 160, szTextOutput, lstrlen(szTextOutput));

if(nObjectShape == 907)
    lstrcpy((LPSTR)szTextOutput, "Object Shape: Car");
else
    lstrcpy((LPSTR)szTextOutput, "Object Shape: Circle");
TextOut(hdc, 10, 180, szTextOutput, lstrlen(szTextOutput));

if(nDrawCL == 909)
    lstrcpy((LPSTR)szTextOutput, (LPSTR)"Center Line: Yes");
else
    lstrcpy((LPSTR)szTextOutput, (LPSTR)"Center Line: No");
TextOut(hdc, 10, 200, szTextOutput, lstrlen(szTextOutput));

switch(nETime)
{
case 801:
    lstrcpy((LPSTR)szTextOutput, (LPSTR)"Experiment Time: 1 min.");
    break;

case 802:
    lstrcpy((LPSTR)szTextOutput, (LPSTR)"Experiment Time: 2 min.");
    break;

case 803:
    lstrcpy((LPSTR)szTextOutput, (LPSTR)"Experiment Time: 3 min.");
    break;

case 804:
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Experiment Time : 4 min." );
break;

case 805:  
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Experiment Time : 5 min." );
break;
}
TextOut( hdc, 10, 220, szTextOutput, lstrlen( szTextOutput) );

switch( nITime )
{
  case 851:
    lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time : 5 sec." );
    break;
  case 852:
    lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time: 10 sec." );
    break;
  case 853:
    lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time: 15 sec." );
    break;
  case 854:
    lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time: 20 sec." );
    break;
  case 855:
    lstrcpy ((LPSTR)szTextOutput, (LPSTR)"Subject Adaptation Time: 30 sec." );
    break;
}
TextOut( hdc, 10, 240, szTextOutput, lstrlen( szTextOutput) );

if( nCurveDir == 870 )
lstrcpy ((LPSTR)szTextOutput, (LPSTR)"First Curve : Right" );
else
  lstrcpy ((LPSTR)szTextOutput, (LPSTR)"First Curve : Left" );
TextOut(hdc, 10, 260, szTextOutput, lstrlen(szTextOutput));

if (nPreviewSelect == 1101)
    lstrcpy((LPSTR)szTextOutput, (LPSTR)"Preview
Selection: Constant Distance");
else
    lstrcpy((LPSTR)szTextOutput, (LPSTR)"Preview
Selection: Constant Time");
TextOut(hdc, 10, 280, szTextOutput, lstrlen(szTextOutput));

DeleteObject(hCurrentPen);

EndPaint(hWnd, &ps);
ValidateRect(hWnd, NULL);
return 0;

} // Pass unprocessed message to DefMDIChildProc

return DefMDIChildProc(hWnd, message, wParam, lParam);

//the function of result window
//==================================
RstWndProc()
//==================================
long FAR PASCAL RstWndProc(HWND hWnd, UINT message, UINT
wParam, LONG lParam)
{
// variables for painting the screen

HPEN hOldPen, hCurrentPen;
HBRUSH hCurrentBrush, hOldBrush;
HDC hdc, hMemdc;
LOCALHANDLE hRstBuffer;
// handle of data structure
NPRSTBUFFER npRstBuffer;
// a near point to the data structure
PAINTSTRUCT ps;
RECT rect;
static int nTmp, nI, nInt[7], nFloat[7];

// variables for printing out

HBITMAP hPrintBitmap;
DOCINFO di;
int nXPageSize, nYPageSize, nXTime, nYTime,
nSelected;
int nErrorCode;
char szErrorInfo[]="";
FARPROC lpAbortProc, lpPrintDlgProc;

switch (message) {
    case WM_COMMAND:
        if (wParam == IDM_PRINT ) {
            // Get printer DC
            hPrintDC = GetPrinterDC () ;
            lpAbortProc = MakeProcInstance((FARPROC)
                  AbortProc, hInst) ;
            lpPrintDlgProc = MakeProcInstance((FARPROC)
                  PrintDlgProc, hInst) ;
            if( hPrintDC != NULL ) {
                bUserCancels = FALSE;
                EnableWindow( hWnd, FALSE ) ;
                hPrintDlg = CreateDialog (hInst, "AbortDlg",
                        hWnd, lpPrintDlgProc) ;
                SetAbortProc( hPrintDC, lpAbortProc ) ;
                nYPageSize=GetDeviceCaps( hPrintDC, VERTRES );
                nXPageSize=GetDeviceCaps( hPrintDC, HORZRES );
                nXTime = nXPageSize/640;
                nYTime = nYPageSize/480;

                if( nXTime > nYTime )
                    nSelected = nYTime;
                else
                    nSelected = nXTime;

                hdc = GetDC( hWnd );
                hMemdc = CreateCompatibleDC( hdc );
                hPrintBitmap=CreateCompatibleBitmap(hdc,640,480);
                hOldBitmap=SelectObject( hMemdc, hPrintBitmap );
                PatBlt(hMemdc,0,0,640,480,RGB(255,255, 255 ));
                // read data from the local heap
                hRstBuffer = GetWindowWord (hWnd, 0) ;
                npRstBuffer = (NPRSTBUFFER) LocalLock
                        (hRstBuffer) ;
                for( nIndex=0; nIndex<7; nIndex++ ) {
                    nCP[nIndex]=npRstBuffer->nRate[nIndex];
                }
                nObjectShape=npRstBuffer->nOSi
                nRBID=npRstBuffer->nRBID;
                nOffRoadDist1=npRstBuffer->nOffRoadDist1;
nOffRoadDist2=npRstBuffer->nOffRoadDist2;
nOffRoadTime=npRstBuffer->nOffRoadTime;
nRBMean00F=npRstBuffer->nRBMeanF;
nRBMean00I=npRstBuffer->nRBMeanI;
nRBSTD= npRstBuffer->nRBSTD;
nRBSTDF=npRstBuffer->nRBSTDF;
AvgSpeedI= npRstBuffer->nAvgSpeedI;
AvgSpeedF=npRstBuffer->nAvgSpeedF;
E Time=npRstBuffer->nE Time;
IT ime=npRstBuffer->nIT ime;
D rawCL=npRstBuffer->nD rawCL;

LocalUnlock (hRstBuffer) ;
switch( nETime )
{
    case 801:
        nRunTimes=600;
        break;

    case 802:
        nRunTimes=1200;
        break;

    case 803:
        nRunTimes=1800;
        break;

    case 804:
        nRunTimes=2400;
        break;

    case 805:
        nRunTimes=3000;
        break;
}

switch( nITime )
{
    case 851:
        nIgnoreTimes=50;
        break;

    case 852:
        nIgnoreTimes=100;
        break;

    case 853:
        nIgnoreTimes=150;
        break;
}
case 854:
    nIgnoreTimes=200;
    break;

case 855:
    nIgnoreTimes=300;
    break;
}

if( nDrawCL == 909 )
{
    for( nIndex=0; nIndex<7; nIndex++ )
    {
        fCP[nIndex]=(float)(nCP[nIndex])/(float)(nRunTimes-nIgnoreTimes);
        nInt[nIndex]=(int)(fCP[nIndex]*100);
        nFloat[nIndex]=(int)(fCP[nIndex]*100000)-nInt[nIndex]*1000;
    }

    hCurrentBrush = GetStockObject( NULL_BRUSH );
    hCurrentPen = GetStockObject( BLACK_PEN );
    hOldPen=SelectObject( hMemdc, hCurrentPen );
    hOldBrush=SelectObject( hMemdc, hCurrentBrush );
    MoveTo( hMemdc, 77, 10 );
    LineTo( hMemdc, 77, 280 );
    MoveTo( hMemdc, 212, 10 );
    LineTo( hMemdc, 212, 280 );
    for( nIndex = 143; nIndex<146; nIndex++)
    {
        for( nI=0; nI<7; nI++ )
        {
            MoveTo( hMemdc, nIndex, 10+nI*40 );
            LineTo( hMemdc, nIndex, 35+nI*40 );
        }
    }
}

else
{
    for( nIndex=0; nIndex<7; nIndex++ )
    {
        fCP[nIndex]=(float)(nCP[nIndex])/
                    (float)(nRunTimes-nIgnoreTimes);
        nInt[nIndex]=(int)(fCP[nIndex]*100);
        nFloat[nIndex]=(int)(fCP[nIndex]*100000)-nInt[nIndex]*1000;
    }
fCP[3]=fCP[5];
fCP[4]=fCP[6];

hCurrentBrush = GetStockObject( NULL_BRUSH );
hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen=SelectObject( hMemdc, hCurrentPen );
hOldBrush=SelectObject( hMemdc, hCurrentBrush );
MoveTo( hMemdc, 77, 10 );
LineTo( hMemdc, 77, 280 );
MoveTo( hMemdc, 212, 10 );
LineTo( hMemdc, 212, 280 );
}

Rectangle ( hMemdc, 10, 300, 340, 420 );
wsprintf( szTextOutput,
   "Car (C) Position With Respect To Right
   Road Side" );
TextOut( hMemdc, 15, 302, szTextOutput, lstrlen( szTextOutput ));
if( nRBID == 0 )
   wsprintf( szTextOutput,
      "Mean = %4i.%3.3i ft.",(int) (nRBMeanOOI*10+nRBMeanOOF/100)/4,
      (((int) (nRBMeanOOI*10+nRBMeanOOF/100)
      %4)*1000+(int)(nRBMeanOOF%100)*10)/4);
else
   wsprintf( szTextOutput,
      "Mean = -%4i.%3.3i ft.. ( - means left
      side. )", (int) (nRBMeanOOI*10+nRBMeanOOF/100)/4,
      (((int) (nRBMeanOOI*10+nRBMeanOOF/100)
      %4)*1000+(int)(nRBMeanOOF%100)*10)/4);
TextOut( hMemdc, 15, 322, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput,
   "Std. Dev. = %5i.%4.4i ft..",(int)(nRBSTDI)/4,
   (long)((long)((int)(nRBSTDI)%4)*10000+(int)(nRBSTDF))/4);
TextOut( hMemdc, 15, 342, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput,
   "Average Speed = %3.3i.%3.3i mph.",(int)(nAvgSpeedI), (int)(nAvgSpeedF) );
TextOut( hMemdc, 15, 362, szTextOutput, lstrlen( szTextOutput ));
wsprintf(szTextOutput,
    "Off Road Dist. = %3i%3.3i ft .. ",
    (int)(nOffRoadDist1)/4,
    (((int)(nOffRoadDist1)%4)*1000+(int)(nOffRoadDist2))/4);
TextOut(hMemdc, 15, 382, szTextOutput, lstrlen(szTextOutput));

wsprintf(szTextOutput,
    "Off Road Time = %3i.%1.1i sec. ",
    (int)(nOffRoadTime/10),
    (int)(nOffRoadTime%10));
TextOut(hMemdc, 15, 402, szTextOutput, lstrlen(szTextOutput));

// draw histogram charts and calculate the
// percentage of
// car positions

Rectangle(hMemdc, 350, 10, 620, 210);
Rectangle(hMemdc, 350, 220, 620, 420);
MoveTo(hMemdc, 390, 20);
LineTo(hMemdc, 390, 190);
LineTo(hMemdc, 600, 190);
MoveTo(hMemdc, 390, 230);
LineTo(hMemdc, 390, 400);
LineTo(hMemdc, 600, 400);
Ellipse(hMemdc, 388, 39, 392, 41);
Ellipse(hMemdc, 388, 114, 392, 116);
Ellipse(hMemdc, 388, 249, 392, 251);
Ellipse(hMemdc, 388, 324, 392, 326);

SelectObject(hMemdc, hOldPen);
SelectObject(hMemdc, hOldBrush);
DeleteObject(hCurrentPen);
DeleteObject(hCurrentBrush);

hCurrentBrush = CreateSolidBrush(RGB(255,255,0));
hCurrentPen = GetStockObject(BLACK_PEN);
hOldPen = SelectObject(hMemdc, hCurrentPen);
hOldBrush = SelectObject(hMemdc, hCurrentBrush);

if( nDrawCL == 909 )
{
    for(nIndex=0; nIndex<7; nIndex++)
    {
        nTmp=(int)(fCP[nIndex]*100*150/100);
    }
Rectangle( hMemdc, 395+nIndex*30, 190-nTmp, 415+nIndex*30, 190 );
nTmp = 0;
for( nI=0; nI<nIndex+1; nI++ )
{
    nTmp=(int)( fCP[nI]*100*150/100 )+nTmp;
}
Rectangle( hMemdc, 395+nIndex*30, 400-nTmp, 415+nIndex*30, 400 );
wsprintf( szTextOutput, "%1.1i", nIndex+1 );
TextOut( hMemdc, 402+nIndex*30, 192,
        szTextOutput, lstrlen( szTextOutput ) );
TextOut( hMemdc, 402+nIndex*30, 402,
        szTextOutput, lstrlen( szTextOutput ) );
}
wsprintf( szTextOutput, "%1.1i", 1 );
TextOut( hMemdc, 42, 20, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 3 );
TextOut( hMemdc, 107, 20, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 5 );
TextOut( hMemdc, 175, 20, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 7 );
TextOut( hMemdc, 242, 20, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 2 );
TextOut( hMemdc, (40+35), (170-20), szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 4 );
TextOut( hMemdc, (108+35), (170-20), szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 6 );
TextOut( hMemdc, (175+35), (170-20), szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[0], nFloat[0] );
TextOut( hMemdc, 7+7, 40+80, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[2], nFloat[2] );
TextOut( hMemdc, 72+7, 40+80, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[4], nFloat[4] );
TextOut( hMemdc, 140+7, 40+80, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[6], nFloat[6] );
TextOut( hMemdc, 207+7, 40+80, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[1], nFloat[1] );
TextOut( hMemdc, 40+7, 170+80, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[3], nFloat[3] );
TextOut( hMemdc, 108+7, 170+80, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[5], nFloat[5] );
TextOut( hMemdc, 175+7, 170+80, szTextOutput, lstrlen( szTextOutput ));

else
{
    for( nIndex=0; nIndex<5; nIndex++ )
    {
        nTmp=(int)( fCP[nIndex]*100*150/100 );
        Rectangle( hMemdc, 395+nIndex*30, 190-nTmp, 415+nIndex*30, 190 );
        nTmp = 0;
        for( nI=0; nI<nIndex+1; nI++ )
        {
            nTmp=(int)( fCP[nI]*100*150/100 )+nTmp;
        }
        Rectangle( hMemdc, 395+nIndex*30, 400-nTmp, 415+nIndex*30, 400 );
        wsprintf( szTextOutput, "%1.1i", nIndex+1 );
        TextOut( hMemdc, 402+nIndex*30, 192, szTextOutput, lstrlen( szTextOutput ));
        TextOut( hMemdc, 402+nIndex*30, 402, szTextOutput, lstrlen( szTextOutput ));
    }

    wsprintf( szTextOutput, "%1.1i", 1 );
    TextOut( hMemdc, 42, 20, szTextOutput, lstrlen( szTextOutput ));
    wsprintf( szTextOutput, "%1.1i", 5 );
    TextOut( hMemdc, 242, 20, szTextOutput, lstrlen( szTextOutput ));
    wsprintf( szTextOutput, "%1.1i", 2 );
TextOut( hMemdc, (40+35), (170-20), szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%1.1i", 3 );
TextOut( hMemdc, (108+35), 20, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%1.1i", 4 );
TextOut( hMemdc, (175+35), (170-20), szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[0], nFloat[0] );
TextOut( hMemdc, 7+7, 40+80, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[6], nFloat[6] );
TextOut( hMemdc, 207+7, 40+80, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[1], nFloat[1] );
TextOut( hMemdc, 40+7, 170+80, szTextOutput, lstrlen( szTextOutput ));
(nFloat[2]+nFloat[4]+nFloat[3])%1000 );
TextOut( hMemdc, 108+7, 40+80, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[5], nFloat[5] );
TextOut( hMemdc, 175+7, 170+80, szTextOutput, lstrlen( szTextOutput ));
)
wsprintf( szTextOutput, "%1.1i.%1.1i", 0, 5 );
TextOut( hMemdc, 360, 105, szTextOutput, lstrlen( szTextOutput ));
TextOut( hMemdc, 360, 315, szTextOutput, lstrlen( szTextOutput ));
wsprintf( szTextOutput, "%1.1i.%1.1i", 1, 0 );
TextOut( hMemdc, 360, 30, szTextOutput, lstrlen( szTextOutput ));
TextOut( hMemdc, 360, 240, szTextOutput, lstrlen( szTextOutput ));
TextOut( hMemdc, 280, 275, "Unit: %", 7);
TextOut( hMemdc, 400, 20, "p.d.f.", 6);
TextOut( hMemdc, 400, 230, "C.D.F.", 6);

SelectObject( hMemdc, holdPen );
SelectObject( hMemdc, holdBrush );
DeleteObject( hCurrentPen );
DeleteObject( hCurrentBrush );

hCurrentBrush = CreateSolidBrush( RGB(255,255,0) );
hCurrentPen = GetStockObject( BLACK_PEN );
holdPen = SelectObject( hMemdc, hCurrentPen );
h0ldBrush = SelectObject( hMemdc, hCurrentBrush );

if( nDrawCL == 909 )
{
    if( nObjectShape == 907 )
        // draw objct shapes
        {
            Rectangle ( hMemdc, 7+23, 54, 7+23+29, 54+47);
            Rectangle ( hMemdc, 72+23, 54, 72+23+29, 54+47);
            Rectangle ( hMemdc, 140+23, 54, 140+23+29, 54+47);
            Rectangle ( hMemdc, 207+23, 54, 207+23+29, 54+47);
            Rectangle ( hMemdc, 40+23, 184, 40+23+29, 184+47);
            Rectangle ( hMemdc, 108+23, 184, 108+23+29, 184+47);
            Rectangle ( hMemdc, 175+23, 184, 175+23+29, 184+47);
        }
    else
    {
        Ellipse ( hMemdc, 7+14, 40+14, 7+23+29, 40+23+29 );
        Ellipse ( hMemdc, 72+14, 40+14, 72+23+29, 40+23+29 );
        Ellipse ( hMemdc, 140+14, 40+14, 140+23+29, 40+23+29 );
        Ellipse ( hMemdc, 207+14, 40+14, 207+23+29, 40+23+29 );
        Ellipse ( hMemdc, 40+14, 170+14, 40+23+29, 170+23+29 );
        Ellipse ( hMemdc, 108+14, 170+14, 108+23+29, 170+23+29 );
        Ellipse ( hMemdc, 175+14, 170+14, 175+23+29, 170+23+29 );
    }
}
else
{
    if( nObjectShape == 907 )
        // draw objct shapes
{  
  Rectangle ( hMemdc, 7+23, 54, 7+23+29, 54+47);  
  Rectangle ( hMemdc, 207+23, 54, 207+23+29, 54+47);  
  Rectangle ( hMemdc, 40+23, 184, 40+23+29, 184+47);  
  Rectangle ( hMemdc, 108+23, 54, 108+23+29, 184+47);  
  Rectangle ( hMemdc, 175+23, 184, 175+23+29, 184+47);  
}  
else  
{  
  Ellipse ( hMemdc, 7+14, 40+14, 7+23+29, 40+23+29 );  
  Ellipse ( hMemdc, 207+14, 40+14, 207+23+29, 40+23+29 );  
  Ellipse ( hMemdc, 40+14, 170+14, 40+23+29, 170+23+29 );  
  Ellipse ( hMemdc, 108+14, 40+14, 108+23+29, 40+23+29 );  
  Ellipse ( hMemdc, 175+14, 170+14, 175+23+29, 170+23+29 );  
}  
}  
SelectObject( hMemdc, hOldBrush );  
DeleteObject( hCurrentBrush );  
SelectObject( hMemdc, hOldPen );  
DeleteObject( hCurrentPen );  

di.cbSize = sizeof(DOCINFO);  
di.lpszDocName="Result";  
di. lpszOutput=NULL;  

nErrorCode=StartDoc( hPrintDC, &di);  

// error code  
if( nErrorCode > 0 )  
{  
  StretchBlt( hPrintDC, 0, 0, nSelected*640,  
              hSelected*480,  
              hMemdc, 0, 0, 640, 480, SRCCOPY );  
  nErrorCode=Escape( hPrintDC, NEWFRAME, 0,  
                     NULL, NULL );  
  if( nErrorCode > 0 )  
  {  
      Escape( hPrintDC, ENDDOC, strlen("Result"),  

"Result", NULL );

else
{
    wsprintf( szErrorInfo, "Printing Error! 
            Error No. = %i ", nErrorCode );
    BWCCMessageBox( hWnd, szErrorInfo, " Print 
            Error", MB_ICONSTOP | MB_OK );
}
else
{
    wsprintf( szErrorInfo, "Printing Error! Error 
            No. = %i ", nErrorCode );
    BWCCMessageBox( hWnd, szErrorInfo, " Print 
            Error", MB_ICONSTOP | MB_OK );
}

if( !bUserCancels )
{
    EnableWindow( hWnd, TRUE );
    DestroyWindow( hPrintDlg);
    SelectObject( hMemdc, hOldBitmap );
    DeleteObject( hPrintBitmap);
    DeleteDC( hMemdc );
    ReleaseDC( hWnd, hdc );
} else
    BWCCMessageBox( hWnd, 
            "Can not find assigned printer!", 
            " Print Error", 
            MB_ICONSTOP | MB_OK );

FreeProcInstance (lpPrintDlgProc);
FreeProcInstance (lpAbortProc);
DeleteDC(hPrintDC);

break;

case WM_CREATE:
    // Allocate memory for window private data
    hRstBuffer = LocalAlloc (LMEM_MOVEABLE |
                           LMEM_ZEROINIT, 
                           sizeof (DATABUFFER)) ;
    npRstBuffer = (NPRSTBUFFER) LocalLock 
                   (hRstBuffer) ;
for( nIndex=0; nIndex<7; nIndex++ )
{
    npRstBuffer->nRate[nIndex] = nCP[nIndex];
}
npRstBuffer->nOS = nObjectShape;
npRstBuffer->nRBID = nRBID;
npRstBuffer->nRBMeanI = nRBMean00I;
npRstBuffer->nRBMeanF = nRBMean00F;
npRstBuffer->nRBSTD = nRBSTD;
npRstBuffer->nRBSTD = nRBSTD;
npRstBuffer->nAvgSpeedI = nAvgSpeedI;
npRstBuffer->nAvgSpeedF = nAvgSpeedF;
npRstBuffer->nETime = nETime;
npRstBuffer->nITime = nITime;
npRstBuffer->nOffRoadDist1 = nOffRoadDist1;
npRstBuffer->nOffRoadDist2 = nOffRoadDist2;
npRstBuffer->nOffRoadTime = nOffRoadTime;
npRstBuffer->nDrawCL = nDrawCL;
LocalUnlock (hRstBuffer);
SetWindowWord (hWnd, 0, hRstBuffer);

// Save some window handles
hdc=GetDC( hWnd );
hCurrentPen = GetStockObject( BLACK_PEN );
hCurrentBrush = GetStockObject( NULL_BRUSH );
hOldPen = SelectObject( hdc, hCurrentPen );
hOldBrush = SelectObject( hdc, hCurrentBrush );
ReleaseDC( hWnd, hdc );
Rectangle( hdc, 370, 10, 620, 210 );
Rectangle( hdc, 370, 220, 620, 420 );
hWndClient = GetParent (hWnd);
hWndFrame = GetParent (hWndClient);
return 0 ;
case WM_DESTROY:
    hdc=GetDC( hWnd );
    SelectObject( hdc, hOldPen );
    DeleteObject( hCurrentPen );
    SelectObject( hdc, hOldBrush );
    DeleteObject( hCurrentBrush );
    ReleaseDC( hWnd, hdc );
    hRstBuffer = GetWindowWord (hWnd, 0 );
LocalFree(hRstBuffer);
return 0;

case WM_PAINT:

    // Paint the window

    hdc = BeginPaint(hWnd, &ps);

    hRstBuffer = GetWindowWord(hWnd, 0);
    npRstBuffer = (NPRSTBUFFER) LocalLock(hRstBuffer);
    for(nIndex=0; nIndex<7; nIndex++)
    {
        nCP[nIndex]=npRstBuffer->nRate[nIndex];
    }
    nObjectShape=npRstBuffer->nOS;
    nRBID=npRstBuffer->nRBID;
    nRBMean00I=npRstBuffer->nRBMeanI;
    nRBMean00F=npRstBuffer->nRBMeanF;
    nRBSTD1=npRstBuffer->nRBSTD1;
    nRBSTDF=npRstBuffer->nRBSTDF;
    nAvgSpeedI=npRstBuffer->nAvgSpeedI;
    nAvgSpeedF=npRstBuffer->nAvgSpeedF;
    nETime=npRstBuffer->nETime;
    nITime=npRstBuffer->nITime;
    nOffRoadDist1=npRstBuffer->nOffRoadDist1;
    nOffRoadDist2=npRstBuffer->nOffRoadDist2;
    nOffRoadTime=npRstBuffer->nOffRoadTime;
    nDrawCL=npRstBuffer->nDrawCL;

    LocalUnlock(hRstBuffer);
    switch(nETime)
    {
        case 801:
            nRunTimes=600;
            break;

        case 802:
            nRunTimes=1200;
            break;

        case 803:
            nRunTimes=1800;
            break;

        case 804:
            nRunTimes=2400;
            break;
case 805:
    nRunTimes=3000;
    break;
}

switch( nITime )
{
    case 851:
        nIgnoreTimes=50;
        break;
    case 852:
        nIgnoreTimes=100;
        break;
    case 853:
        nIgnoreTimes=150;
        break;
    case 854:
        nIgnoreTimes=200;
        break;
    case 855:
        nIgnoreTimes=300;
        break;
}

if( nDrawCL == 909 )
{
    for( nIndex=0; nIndex<7; nIndex++ )
    {
        fCP[nIndex]=(float)(nCP[nIndex])/(float)(nRunTimes-
nIgnoreTimes);
        nInt[nIndex]=(int)(fCP[nIndex]*100);
        nFloat[nIndex]=(int)(fCP[nIndex]*100000)-
        nInt[nIndex]*1000;
    }

    if( nObjectShape == 907 )
    { 
        hBitmap1 = LoadBitmap ( hInst, "CABITMAP" );
    } else
    { 
        hBitmap1 = LoadBitmap ( hInst, "CIBITMAP" );
    }

    hMemdc = CreateCompatibleDC ( hdc );
    SelectObject ( hMemdc, hBitmap1 );
    StretchBlt ( hdc,7,40,75,75,hMemdc,0,
0,75,75,SRCCOPY);
StretchBlt ( hdc,72,40,75,75,hMemdc,0,
0,75,75,SRCCOPY);
StretchBlt ( hdc,140,40,75,75,hMemdc,0,
0,75,75,SRCCOPY);
StretchBlt ( hdc,207,40,75,75,hMemdc,0,
0,75,75,SRCCOPY);
hCurrentBrush = GetStockObject( NULL_BRUSH );
hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen=SelectObject( hdc, hCurrentPen );
hOldBrush=SelectObject( hdc, hCurrentBrush );
GetClientRect (hWnd, &rect);

MoveTo( hdc, 77, 10 );
LineTo( hdc, 77, 280 );
MoveTo( hdc, 212, 10 )
LineTo( hdc, 212, 280 );

for( nIndex = 143; nIndex<146; nIndex++)
{
    for( nI=0; nI<7; nI++ )
    {
        MoveTo( hdc, nIndex, 10+nI*40 );
        LineTo( hdc, nIndex, 35+nI*40 );
    }
}
StretchBlt ( hdc,40,170,75,75,hMemdc,0,
0,75,75,SRCCOPY);
StretchBlt ( hdc,108,170,75,75,hMemdc,0,
0,75,75,SRCCOPY);
StretchBlt ( hdc,175,170,75,75,hMemdc,0,
0,75,75,SRCCOPY);

DeleteDC ( hMemdc );
DeleteObject ( hBitmap1 );
}
else
{
    for( nIndex=0; nIndex<7; nIndex++ )
    {
        fCP[nIndex]=(float)(nCP[nIndex])/
                    (float)(nRunTimes-nIgnoreTimes);
        nInt[nIndex]=(int)(fCP[nIndex]*100);
        nFloat[nIndex]=(int)(fCP[nIndex]*100000)-
                        nInt[nIndex]*1000;
    }
    fCP[3]=fCP[5];
    fCP[4]=fCP[6];
if( nObjectShape == 907 )
hBitmap1 = LoadBitmap ( hInst, "CABITMAP" );
else
hBitmap1 = LoadBitmap ( hInst, "CIBITMAP" );

hMemdc = CreateCompatibleDC ( hdc );
SelectObject ( hMemdc, hBitmap1 );
StretchBlt ( hdc, 7, 40, 75, 75, hMemdc, 0, 0, 75, 75, SRCCOPY );
StretchBlt ( hdc, 207, 40, 75, 75, hMemdc, 0, 0, 75, 75, SRCCOPY );

hCurrentBrush = GetStockObject( NULL_BRUSH );
hCurrentPen = GetStockObject( BLACK_PEN );
hOldPen=SelectObject( hdc, hCurrentPen );
hOldBrush=SelectObject( hdc, hCurrentBrush );
GetClientRect (hWnd, &rect);
MoveTo( hdc, 77, 10 );
LineTo( hdc, 77, 280 );
MoveTo( hdc, 212, 10 );
LineTo( hdc, 212, 280 );

StretchBlt ( hdc, 40, 170, 75, 75, hMemdc, 0, 0, 75, 75, SRCCOPY );
StretchBlt ( hdc, 108, 40, 75, 75, hMemdc, 0, 0, 75, 75, SRCCOPY );
StretchBlt ( hdc, 175, 170, 75, 75, hMemdc, 0, 0, 75, 75, SRCCOPY );

DeleteDC ( hMemdc );
DeleteObject ( hBitmap1 );
}

Rectangle ( hdc, 10, 300, 340, 420 );
wsprintf( szTextOutput,
"Car (C) Position With Respect To Right Road Side" );
TextOut( hdc, 15, 302, szTextOutput, lstrlen( szTextOutput ) );

if( nRBID == 0 )
wsprintf( szTextOutput,
"Mean = %4i.%3.3i ft.",
(int)(nRBMeanOOI*10+nRBMeanOOF/100)/4,
(((nRBMeanOOI*10+nRBMeanOOF/100)
%4)*1000+(int)(nRBMeanOOF%100)*10)/4);
else
wsprintf( szTextOutput,
"Mean = -%4i.%3.3i ft..( - means left side. )",
(int)(nRBMean00I*10+nRBMean00F/100)/4,
((nRBMean00I*10+nRBMean00F/100)
 %4)*1000+(int)(nRBMean00F%100)*10)/4);
TextOut( hdc, 15, 322, szTextOutput, lstrlen(szTextOutput ));

wsprintf( szTextOutput,
"Std. Dev. = %5i.%4.4i ft.. ",
(int)(nRBSTDf)/4,
(long)((long)((int)(nRBSTDf)%4)*10000
 +(int)(nRBSTDF))/4 );
TextOut( hdc, 15, 342, szTextOutput, lstrlen(szTextOutput ));

wsprintf( szTextOutput,
"Average Speed = %3.3.3i . %3 .3i mph. ",
(int)(nAvgSpeedf), (int)(nAvgSpeedf) );
TextOut( hdc, 15, 362, szTextOutput, lstrlen(szTextOutput ));

wsprintf( szTextOutput,
"Off Road Dist. = %3i%3.3i ft.. ",
(int)(nOffRoadDist1)/4,
(((int)(nOffRoadDist1)%4)*10000
 +(int)(nOffRoadDist2))/4 );
TextOut( hdc, 15, 382, szTextOutput, lstrlen(szTextOutput ));

wsprintf( szTextOutput,
"Off Road Time = %3i.%1.1i sec. ",
(int)(nOffRoadTime/10),
(int)(nOffRoadTime%10) );
TextOut( hdc, 15, 402, szTextOutput, lstrlen(szTextOutput ));

// draw histogram charts and calculate the
// percentage of
// car positions

Rectangle ( hdc, 350, 10, 620, 210 );
Rectangle ( hdc, 350, 220, 620, 420 );
MoveTo( hdc, 390, 20 );
LineTo( hdc, 390, 190 );
LineTo( hdc, 600, 190 );
MoveTo( hdc, 390, 230 );
LineTo( hdc, 390, 400 );
LineTo( hdc, 600, 400 );
Ellipse ( hdc, 388, 39, 392, 41 );
Ellipse ( hdc, 388, 114, 392, 116 );
Ellipse ( hdc, 388, 249, 392, 251 );
Ellipse( hdc, 388, 324, 392, 326 );
wsprintf( szTextOutput, "%1.1i.%1.1i", 0, 5 );
TextOut( hdc, 360, 105, szTextOutput, lstrlen(szTextOutput));
TextOut( hdc, 360, 315, szTextOutput, lstrlen(szTextOutput));
wsprintf( szTextOutput, "%1.1i.%1.1i", 1, 0 );
TextOut( hdc, 360, 30, szTextOutput, lstrlen(szTextOutput));
TextOut( hdc, 360, 240, szTextOutput, lstrlen(szTextOutput));

TextOut( hdc, 280, 275, "Unit: %", 7);
TextOut( hdc, 400, 20, "p.d.f.", 6);
TextOut( hdc, 400, 230, "C.D.F.", 6);

DeleteObject( hCurrentBrush );
hCurrentBrush = CreateHatchBrush( HS_DIAGCROSS, RGB(255,0,0));
SelectObject( hdc, hCurrentBrush );

if( nDrawCL == 909 )
{
    for( nIndex=0; nIndex<7; nIndex++ )
    {
        nTmp=(int)( fCP[nIndex]*100*150/100 );
        Rectangle( hdc, 395+nIndex*30, 190-nTmp, 
                    415+nIndex*30, 190 );
        wsprintf( szTextOutput, "%1.1i", nIndex+1 );
        TextOut( hdc, 402+nIndex*30, 192, szTextOutput, 
                 lstrlen(szTextOutput));
        TextOut( hdc, 402+nIndex*30, 402, szTextOutput, 
                 lstrlen(szTextOutput));
    }
    nTmp = 0;
    for( nI=0; nI<nIndex+1; nI++ )
    {
        nTmp=(int)( fCP[nI]*100*150/100 )+nTmp;
    }
    Rectangle( hdc, 395+nIndex*30, 400-nTmp, 
                415+nIndex*30, 400 );
    }
    wsprintf( szTextOutput, "%1.1i", 1 );
    TextOut( hdc, 42, 20, szTextOutput, lstrlen(szTextOutput));
    wsprintf( szTextOutput, "%1.1i", 3 );
TextWriter( hdc, 107, 20, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%1.1i", 5);
TextWriter( hdc, 175, 20, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%1.1i", 7);
TextWriter( hdc, 242, 20, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%1.1i", 2);
TextWriter( hdc, (40+35), (170-20), szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%1.1i", 4);
TextWriter( hdc, (108+35), (170-20), szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%1.1i", 6);
TextWriter( hdc, (175+35), (170-20), szTextOutput, lstrlen(szTextOutput ));

wsprintf(szTextOutput, "%3.2i.%3.3i", nInt[0], nFloat[0]);
TextWriter( hdc, 7+7, 40+80, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%3.2i.%3.3i", nInt[2], nFloat[2]);
TextWriter( hdc, 72+7, 40+80, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%3.2i.%3.3i", nInt[4], nFloat[4]);
TextWriter( hdc, 140+7, 40+80, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%3.2i.%3.3i", nInt[6], nFloat[6]);
TextWriter( hdc, 207+7, 40+80, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%3.2i.%3.3i", nInt[1], nFloat[1]);
TextWriter( hdc, 40+7, 170+80, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%3.2i.%3.3i", nInt[3], nFloat[3]);
TextWriter( hdc, 108+7, 170+80, szTextOutput, lstrlen(szTextOutput ));
wsprintf(szTextOutput, "%3.2i.%3.3i", nInt[5], nFloat[5]);
TextWriter( hdc, 175+7, 170+80, szTextOutput, lstrlen(szTextOutput ));
}
else
for( nIndex=0; nIndex<5; nIndex++ )
{
    nTmp=(int)( fCP[nIndex]*100*150/100 );
    Rectangle( hdc, 395+nIndex*30, 190-nTmp,
               415+nIndex*30, 190 );
    wsprintf( szTextOutput, "%1.1i", nIndex+1 );
    TextOut( hdc, 402+nIndex*30, 192, szTextOutput,
             lstrlen( szTextOutput ) );
    TextOut( hdc, 402+nIndex*30, 402, szTextOutput,
             lstrlen( szTextOutput ) );
}

nTmp = 0;
for( nI=0; nI<nIndex+1; nI++ )
{
    nTmp=(int)( fCP[nI]*100*150/100 )+nTmp;
}
    Rectangle( hdc, 395+nIndex*30, 400-nTmp,
               415+nIndex*30, 400 );
}

wsprintf( szTextOutput, "%1.1i", 1 );
TextOut( hdc, 42, 20, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 5 );
TextOut( hdc, 242, 20, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 2 );
TextOut( hdc, (40+35), (170-20), szTextOutput,
          lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 3 );
TextOut( hdc, (108+35), 20, szTextOutput,
          lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%1.1i", 4 );
TextOut( hdc, (175+35), (170-20), szTextOutput,
          lstrlen( szTextOutput ) );

wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[0], nFloat[0] );
TextOut( hdc, 7+7, 40+80, szTextOutput, lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[6], nFloat[6] );
TextOut( hdc, 207+7, 40+80, szTextOutput,
          lstrlen( szTextOutput ) );
wsprintf( szTextOutput, "%3.2i.%3.3i", nInt[1], nFloat[1] );
TextOut( hdc, 40+7, 170+80, szTextOutput,
          lstrlen( szTextOutput ) );
(nFloat[2]+nFloat[4]+nFloat[3]/1000,
(nFloat[2]+nFloat[4]+nFloat[3])%1000);
TextOut( hdc, 108+7, 40+80, szTextOutput,
1strlen(szTextOutput ));
wsprintf(szTextOutput, "%3.2i.%3.3i", nInt[5],
nFloat[5]);
TextOut( hdc, 175+7, 170+80, szTextOutput,
1strlen(szTextOutput ));
}
SelectObject(hdc, hOldPen);
SelectObject(hdc, hOldBrush);
DeleteObject(hCurrentPen);
DeleteObject(hCurrentBrush);
EndPaint(hWnd, &ps);
ValidateRect(hWnd, NULL);
return 0;

} // Pass unprocessed message to DefMDIChildProc

return DefMDIChildProc(hWnd, message, wParam, lParam);

}

//=====================================================================
//Dial1Proc()
//=====================================================================
// Users can choose which data segemnet to look.
BOOL FAR PASCAL Dial1Proc(hWnd, message, wParam, lParam)
HWND hWnd;
UINT message;
UINT wParam;
LONG lParam;
{
switch (message)
{
    case WM_INITDIALOG:
        CheckRadioButton(hWnd, IDD_ALLSEG, IDD_5SEG,
nSEGBUFFER);
        SetDlgItemInt(hWnd, IDD_EXPTIME, nETime-800, TRUE);
        return(TRUE);
    case WM_COMMAND:
        switch (wParam)
        {
        case IDCANCEL:


EndDialog ( hDlg, FALSE );
break;

case IDOK :
switch(nSEGTmp)
{
    case 1:
nSEG = IDD_1SEG;
    break;
    
case 2:
nSEG = IDD_2SEG;
    break;
    
case 3:
nSEG = IDD_3SEG;
    break;
    
case 4:
nSEG = IDD_4SEG;
    break;
    
case 5:
nSEG = IDD_5SEG;
    break;
    
case 0:
nSEG = IDD_ALLSEG;
    break;
}

nSEGBUFFER = nSEG;
if( nSEG-700 <= nETime-800)
    EndDialog ( hDlg, TRUE );
else
    BWCCMessageBox( hDlg, "Input Error.",
                    "Error!", MB_ICONHAND | MB_OK );
break;

case IDD_ALLSEG :
nSEGTmp = 0;
    break;

case IDD_1SEG :
nSEGTmp = 1;
    break;

case IDD_2SEG :
nSEGTmp = 2;
break;

case IDD_3SEG :
    nSEGTmp = 3;
    break;

case IDD_4SEG :
    nSEGTmp = 4;
    break;

case IDD_5SEG :
    nSEGTmp = 5;
    break;

default :  return ( FALSE );
} break;

default:
    return ( FALSE );
} return ( TRUE );

////////////////////////////////////////////////////////////
//                          Dial2Proc()
////////////////////////////////////////////////////////////
// The dialog box is used to open *.dat files.

BOOL FAR PASCAL Dial2Proc ( hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    switch ( message )
    {
    case WM_INITDIALOG :
        DlgDirList( hDlg, "\*.dat", IDD_DIRECTOR, IDD_PATH,
                   0x4030 );
        SetDlgItemText ( hDlg, IDD_FILENAME, "\*.dat" );
        return ( TRUE );

    case WM_COMMAND :
        switch ( wParam )
        {
        case IDCANCEL :
            EndDialog ( hDlg, FALSE );
        break;
case IDOK :  // get file name and path
    GetDlgItemText( hDlg, IDD_PATH, (LPSTR)szPathTmp, 69 );
    GetDlgItemText( hDlg, IDD_FILENAME, (LPSTR)szFileName, 79 );
    EndDialog ( hDlg, TRUE );
    break;

case IDD_DIRECTOR :
    if( HIWORD( lParam ) == 2 )  // double click
        {  // find the directory
            SendDlgItemMessage( hDlg, IDD_DIRECTOR, LB_GETCURSEL, 0, 0L );
            DlgDirSelect( hDlg, (LPSTR)szDirectory, IDD_DIRECTOR );
            lstrcpy((LPSTR)szDirectoryTmp, (LPSTR)szDirectory);
            lstrcat((LPSTR)szDirectoryTmp, "*.dat");
            DlgDirList( hDlg, szDirectoryTmp, IDD_DIRECTOR, IDD_PATH, 0x4030 );
            DlgDirSelect( hDlg, (LPSTR)szDirectory, IDD_DIRECTOR );
            SetDlgItemText( hDlg, IDD_FILENAME, (LPSTR)szDirectory );
        }
    break;

default :  return ( FALSE );
}  break;

default:
    return ( FALSE );
}  return ( TRUE );

//====================================================================
// Dia13Proc()
//====================================================================
// The user can select display mode in this dialog box, text
// or graphic.

BOOL FAR PASCAL Dia13Proc ( hDlg, message, wParam, lParam )
HWND hDlg;
UINT message;
UINT wParam;
LONG lParam;
{
    switch ( message )
    {
        case WM_INITDIALOG :
            CheckRadioButton ( hDlg, 910, 920, nDisplayMode );
            return ( TRUE );
        case WM_COMMAND :
            switch ( wParam )
            {
                case IDCANCEL :
                    EndDialog ( hDlg, FALSE );
                    break;
                case IDOK :
                    switch ( nDisplayModeTemp )
                    {
                        case 0:
                            nDisplayMode = 910;
                            break;
                        case 1:
                            nDisplayMode = 920;
                            break;
                    }
                    EndDialog ( hDlg, TRUE );
                    break;
                case 910:
                    nDisplayModeTemp = 0;
                    break;
                case 920:
                    nDisplayModeTemp = 1;
                    break;
                default: return ( FALSE );
            }
            break;
        default:
            return ( FALSE );
    }
    return ( TRUE );
}
The dialog box is used to open *.inf files.

BOOL FAR PASCAL Dial4Proc ( hWnd hDlg, UINT message, UINT wParam, LONG lParam )
{
    switch ( message )
    {
        case WM_INITDIALOG :
            DlgDirList( hDlg, "*.inf", IDD_DIRECTOR, IDD_PATH, Ox4030 );
            SetDlgItemText ( hDlg, IDD_FILENAME, "*.inf" );
            return ( TRUE );

        case WM_COMMAND :
            switch ( wParam )
            {
                case IDCANCEL : // if cancel button is pushed
                    EndDialog ( hDlg, FALSE );
                    break;

                case IDOK :
                    // get file name and path

                    GetDlgItemText( hDlg, IDD_PATH, (LPSTR)szPathTmp, 69 );
                    GetDlgItemText( hDlg, IDD_FILENAME, (LPSTR)szFileName, 79 );
                    EndDialog ( hDlg, TRUE );
                    break;

                case IDD_DIRECTOR :
                    if ( HIWORD( lParam ) == 2 ) // double click
                        
                        // find the directory

                        SendDlgItemMessage( hDlg, IDD_DIRECTOR, LB_GETCURSEL, 0, 0L );
                        DlgDirSelect( hDlg, (LPSTR)szDirectory, IDD_DIRECTOR );
                        lstrcpy((LPSTR)szDirectoryTmp,
                                (LPSTR)szDirectory);  
                        lstrcat((LPSTR)szDirectoryTmp,
                                "*.inf" );
The dialog box is used to open *.rst files.

BOOL FAR PASCAL DialSProc ( hWnd, message, wParam, lParam )

HWND hWnd;
UINT message;
UINT wParam;
LONG lParam;

switch ( message )
{
case WM_INITDIALOG :

    // set up default value
    
        DlgDirList( hWnd, "\*.rst", IDD_DIRECTOR, IDD_PATH, 0x4030 );
        SetDlgItemText ( hWnd, IDD_FILENAME, "\*.rst" );
    return ( TRUE );

case WM_COMMAND :
    switch ( wParam )
    {
        case IDCANCEL :
            EndDialog ( hWnd, FALSE );
            break;

        case IDOK :


// get file name and path
GetDlgItemText( hDlg, IDD_PATH, (LPSTR)szPathTmp,
69 );
GetDlgItemText( hDlg, IDD_FILENAME,
(LPSTR)szFileName, 79 );
EndDialog ( hDlg, TRUE );
break;

case IDD_DIRECTOR :
if( HIWORD( lParam ) == 2 ) // double click
{
    // find the directory
    SendDlgItemMessage( hDlg, IDD_DIRECTOR,
LB_GETCURSEL, 0, 0L );
    DlgDirSelect( hDlg, (LPSTR)szDirectory,
IDD_DIRECTOR );
lstrcpy((LPSTR)szDirectoryTmp,
(LPSTR)szDirectory);
lstrcat((LPSTR)szDirectoryTmp, ".*.rst");
    DlgDirList( hDlg, szDirectoryTmp, IDD_DIRECTOR,
IDD_PATH, 0x4030 );
    DlgDirSelect( hDlg, (LPSTR)szDirectory,
IDD_DIRECTOR );
    SetDlgItemText( hDlg, IDD_FILENAME,
(LPSTR)szDirectory );
}
break;

default : return ( FALSE );
}
break;

default:
    return ( FALSE );
}
return ( TRUE );
}

strconv-------------------------------------------
// GetValue()
// the subroutine is designed to get integer value.
int GetValue( nDigit )
int nDigit;
{    int nReturn;
llseek(hFileHandle, nDigit, 0);
llseek(hFileHandle, (LPSTR)sz100, 1);
llseek(hFileHandle, nDigit+1, 0);
llseek(hFileHandle, (LPSTR)sz10, 1);
llseek(hFileHandle, nDigit+2, 0);
lread(hFileHandle, (LPSTR)sz1, 1);

n100=(int)(*sz100)-48;
n10=(int)(*sz10)-48;
n1=(int)(*sz1)-48;
nReturn=n100*100+n10*10+n1;
return nReturn;

//==================================================================================
// GetPrinterDC()
//==================================================================================
// To find the current printer driver

HDC GetPrinterDC (void)
{
    static char szPrinter [80] ;
    char *szDevice, *szDriver, *szOutput ;

    // find printer device from win.ini
    GetProfileString ("windows", "device", ",", 
    szPrinter, 80) ;

    if (NULL != (szDevice = strtok (szPrinter, "," )) &&
        NULL != (szDriver = strtok (NULL, "," )) &&
        NULL != (szOutput = strtok (NULL, "," )))

        return CreateDC (szDriver, szDevice, szOutput, NULL) ;

    return 0 ;
}

//==================================================================================
// AbortProc()
//==================================================================================
// The dialog box is a modeless dialog box. It allows the user to
// cancel the print job.

int FAR PASCAL AbortProc (HDC hPrintDC, int nCode)
{
    MSG msg ;
while (!bUserCancels && PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))
{
    if (!hPrintDlg || !IsDialogMessage (hPrintDlg, &msg))
    {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
}
return (!bUserCancels);

//------------------------------------------------------------------------------------------------------------------------
PrintDlgProc()
//------------------------------------------------------------------------------------------------------------------------
// The dialog box will allow users to cancel printing jobs.

int FAR PASCAL PrintDlgProc (HWND hDlg, UINT message, UINT wParam,
LONG lParam)
{
    switch (message)
    {
    case WM_INITDIALOG:
        EnableMenuItem (GetSystemMenu (hDlg, FALSE), SC_CLOSE, MF_GRAYED);
        // Don't allow the subject close the dialog box
        // from system menu

        return TRUE ;

    case WM_COMMAND:
        bUserCancels = TRUE ;
        EnableWindow (GetParent (hDlg), TRUE) ;
        DestroyWindow (hDlg) ;
        hPrintDlg = 0 ;
        return TRUE ;
    }
return FALSE ;
}
Appendix C

Resource Definition File (SIMU.RC) of SIMULATOR

with Detailed Description
//===========================================================
// SIMM.RC resource script
//===========================================================

#include "windows.h"
#include "simm.h"

// The main menu is named SimmMenu

SimmMenu MENU

BEGIN
POPUP "&Options"
BEGIN
MENUITEM "&New Subject...", IDM_NEW
MENUITEM SEPARATOR
MENUITEM "E&xit", IDM_EXIT
END

POPUP "\a&About"
BEGIN
MENUITEM "About &Simmulator...", IDM_ABOUT
END
END

// The standard structure of resources in .RC file is
// NAME TYPE DATA

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>THESISICON</td>
<td>ICON</td>
<td>&quot;thesis.ico&quot;</td>
</tr>
<tr>
<td>CARBITMAP</td>
<td>BITMAP</td>
<td>&quot;car1.bmp&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>// used in animation</td>
</tr>
<tr>
<td>CIRCLEBITMAP</td>
<td>BITMAP</td>
<td>&quot;cir3.bmp&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>// used in animation</td>
</tr>
<tr>
<td>CABITMAP</td>
<td>BITMAP</td>
<td>&quot;car.bmp&quot;</td>
</tr>
<tr>
<td>box</td>
<td></td>
<td>// used in dialog</td>
</tr>
<tr>
<td>CIBITMAP</td>
<td>BITMAP</td>
<td>&quot;cir.bmp&quot;</td>
</tr>
<tr>
<td>box</td>
<td></td>
<td>// used in dialog</td>
</tr>
<tr>
<td>CRBITMAP</td>
<td>BITMAP</td>
<td>&quot;cr.bmp&quot;</td>
</tr>
<tr>
<td>window</td>
<td></td>
<td>// for copyright</td>
</tr>
</tbody>
</table>

// This dialog box is used to show information of SIMULATOR.

Aboutbox DIALOG 78, 44, 142, 92
STYLE WS_DLGFRAME | WS_POPUP | WS_CAPTION | WS_BORDER
CLASS "BorDlg"     // Borland Customer Style Dialog Box
CAPTION "About Simmulator..."
BEGIN
CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD |
WS_VISIBLE | WS_TABSTOP, 99, 110, 32, 20 |
RTEXT "Ohio University", -1, 165, 110, 56, 8, SS_RIGHT |
WS_CHILD | WS_VISIBLE | WS_GROUP |
ICON "ThesisICON", -1, 103, 41, 16, 16, WS_CHILD |
WS_VISIBLE |
LTEXT "Industrial and Systems Engineering", -1, 101, 99, 120, 8, WS_CHILD | WS_VISIBLE | WS_GROUP |
RTEXT "June/1994", -1, 185, 120, 36, 7, SS_RIGHT |
WS_CHILD | WS_VISIBLE | WS_GROUP |
CONTROL "SIMULATOR 1.0", -1, "STATIC", SS_CENTER |
SS_NOPREFIX | WS_CHILD | WS_VISIBLE | WS_GROUP, 87, 10, 59, 8 |
CONTROL "", 102, "BorShade", 32769 | WS_CHILD |
WS_VISIBLE, 76, 7, 80, 13 |
CONTROL "", 104, "BorShade", 32770 | WS_CHILD |
WS_VISIBLE, 2, 23, 221, 113 |
CONTROL "", 103, "BorShade", 32769 | WS_CHILD |
WS_VISIBLE, 101, 39, 20, 20 |
LTEXT "Ergonomics Laboratory", -1, 100, 90, 109, 8, WS_CHILD | WS_VISIBLE | WS_GROUP |
LTEXT "Thesis Advisor :", -1, 130, 28, 54, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP |
LTEXT "Dr. Helmut T. Zwahlen", -1, 130, 38, 91, 8, WS_CHILD | WS_VISIBLE | WS_GROUP |
LTEXT "Programmer :", -1, 130, 49, 54, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP |
LTEXT "Jui-Lin Chen", -1, 130, 59, 91, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP |
CONTROL "", 105, "BorShade", 3 | WS_CHILD | WS_VISIBLE, 93, 23, 1, 113 |
LTEXT "PREVIEW TRACKING", -1, 6, 31, 79, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP |
LTEXT "TASK SIMULATOR AND", -1, 7, 47, 79, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP |
LTEXT "ANALYSIS PACKAGE", -1, 7, 63, 79, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP |
LTEXT "by Jui-Lin Chen And", -1, 8, 101, 79, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP |
LTEXT "Helmut T. Zwahlen", -1, 8, 112, 78, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP |
LTEXT "251 copyrights 1994", -1, 7, 90, 81, 8, SS_CENTER |
SS_NOPREFIX | WS_CHILD | WS_VISIBLE | WS_GROUP |
LTEXT "Human Factors And " | WS_CHILD | WS_VISIBLE, 7, 89, 81, 34 |
END
This dialog box is used to set preview, initial seed, and road width.

Dial1 DIALOG 30, 14, 257, 147
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CLASS "BorDlg"
CAPTION "Variables Setup 1"
BEGIN
    CONTROL "Fixed time", IDD_FT, "BUTTON", WS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE |
    WS_GROUP | WS_TABSTOP, 20, 19, 49, 12
    CONTROL "Fixed Dist", IDD_FD, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE,
    87, 19, 49, 12
    LTEXT "Initial Seed", -1, 164, 21, 42, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
    CONTROL "0", IDD_SEED, "EDIT", ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 211, 19, 29, 12
    CONTROL "Random Number", -1, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 156, 5, 94, 67
    CONTROL "Road Width", -1, "button", BS_GROUPBOX | WS CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 211, 19, 29, 12
    LTEXT "Road Width", -1, 161, 94, 39, 8, WS CHILD | WS_VISIBLE | WS_GROUP
    CONTROL "50", IDD_ROADWIDTH, "EDIT", ES_LEFT | WS_CHILD |
    WS_VISIBLE | WS_BORDER | WS_TABSTOP, 206, 92, 16, 12
    LTEXT "ft.", -1, 225, 94, 7, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
    CONTROL "OK", IDOK, "BorBtn", BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 79, 122, 32, 19
    CONTROL "CANCEL", IDCANCEL, "BorBtn", BS_AUTOCHECKBOX |
    WS_CHILD | WS_VISIBLE | WS_TABSTOP, 156, 123, 32, 20
    LTEXT "The initial seed that used ", -1, 163, 35, 76, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
    LTEXT "used in last experiment ", -1, 163, 46, 77, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
    LTEXT "is", -1, 163, 59, 12, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
    LTEXT ".", IDD_SEEDSHOW, 175, 59, 19, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
    LTEXT ".", -1, 194, 59, 2, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
This dialog box is created to input file name to save data.

Dial2 DIALOG 88, 49, 146, 108
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CLASS "BorDlg"
CAPTION "Save..."
BEGIN
EDITTEXT IDD_FILENAME, 86, 6, 37, 12, ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP
END
This dialog box is used to set the relative frequency of curve, time lag, input control gain, center line and object shape.
SCROLLBAR IDD_GAINS, 8, 123, 74, 9, SBS_HORZ | WS_CHILD | WS_VISIBLE
CONTROL "Input Gain", 102, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 3, 100, 116, 36
EDITTEXT IDD_GAINE, 88, 118, 17, 12, ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP
LTEXT "1-7", -1, 93, 108, 12, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "0-100", -1, 94, 10, 20, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "0", -1, 13, 13, 5, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "100", -1, 75, 13, 12, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "1000", -1, 73, 61, 16, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "0", -1, 12, 61, 5, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "1", -1, 10, 111, 5, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "7", -1, 76, 111, 5, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
CONTROL "Car", IDD_SHAPECAR, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 130, 17, 29, 12
CONTROL "Circle", IDD_SHAPECIRCLE, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 130, 35, 30, 12
CONTROL "Object Shape", 104, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 124, 3, 80, 50
CONTROL ",", -1, "static", SS_BLACKFRAME | WS_CHILD | WS_VISIBLE, 168, 16, 33, 33
CONTROL "Center Line", 104, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 125, 60, 80, 47
CONTROL "Yes", IDD_CLYES, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 131, 75, 29, 12
CONTROL "No", IDD_CLNO, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 131, 92, 29, 12
CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 128, 115, 32, 20
CONTROL "CANCEL", IDCANCEL, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 171, 114, 32, 20
// This dialog box is used to input subject ID.

Dial0 DIALOG 82, 55, 155, 29
STYLE DS.MODALFRAME | WS_POPUP | WS_CAPTION
CLASS "BorD1g"
CAPTION "ID of Subject"
BEGIN
  LTEXT "Subject :", -1, 3, 10, 29, 8, WS_CHILD |
  WS_VISIBLE | WS_GROUP
  EDITTEXT IDD_NAME, 37, 8, 76, 12, ES_LEFT | WS_CHILD |
  WS_VISIBLE | WS_BORDER | WS_TABSTOP
  CONTROL "", -1, "BorShade", 3 | WS_CHILD | WS_VISIBLE, 116, 0, 3, 30
  CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD |
  WS_VISIBLE | WS_TABSTOP, 120, 5, 32, 20
END

// This dialog box is used to set experiment time, subject adapt time and the direction of the first curve.

Dial4 DIALOG 83, 32, 169, 149
STYLE DS.MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CLASS "BorD1g"
CAPTION "Variables Setup 3"
BEGIN
  CONTROL "1 min.", IDD_1, "BUTTON", BS_AUTORADIOBUTTON |
  WS_CHILD | WS_VISIBLE | WS_GROUP |
  WS_TABSTOP, 7, 26, 28, 12
  CONTROL "2 min.", IDD_2, "BUTTON", BS_AUTORADIOBUTTON |
  WS_CHILD | WS_VISIBLE, 39, 26, 28, 12
  CONTROL "3 min.", IDD_3, "BUTTON", BS_AUTORADIOBUTTON |
  WS_CHILD | WS_VISIBLE, 71, 26, 28, 12
  CONTROL "4 min.", IDD_4, "BUTTON", BS_AUTORADIOBUTTON |
  WS_CHILD | WS_VISIBLE, 103, 26, 28, 12
  CONTROL "5 min.", IDD_5, "BUTTON", BS_AUTORADIOBUTTON |
  WS_CHILD | WS_VISIBLE, 135, 26, 28, 12
  CONTROL "10 s.", IDD_10, "BUTTON", BS_AUTORADIOBUTTON |
  WS_CHILD | WS_VISIBLE, 39, 64, 28, 12
CONTROL "15 s.", IDD_15, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 71, 64, 28, 12  
CONTROL "20 s.", IDD_20, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 103, 64, 28, 12  
CONTROL "30 s.", IDD_30, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 135, 64, 28, 12  
CONTROL "Subject Adaptation Time", 856, "button",  
   BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 4, 49, 162, 30  
CONTROL "Right", IDD_RIGHTCURVE, "BUTTON",  
   BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE |  
   WS_GROUP | WS_TABSTOP, 18, 103, 33, 12  
CONTROL "Left", IDD_LEFTCURVE, "BUTTON",  
   BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 18, 121, 33, 12  
CONTROL "First Curve", 872, "button", BS_GROUPBOX |  
   WS_CHILD | WS_VISIBLE, 5, 87, 107, 56  
CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD |  
   WS_VISIBLE | WS_TABSTOP, 127, 91, 32, 20  
CONTROL "CANCEL", IDCANCEL, "BorBtn", BS_PUSHBUTTON |  
   WS_CHILD | WS_VISIBLE | WS_TABSTOP, 128, 120, 32, 20  
END  

// This dialog box is used to set up preview time and object  
// speed.  

FPTbox DIALOG 57, 17, 214, 176  
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU  
CLASS "BorDlg"  
CAPTION "Constant Preview Time"  
BEGIN  
CONTROL "Preview Time: sec", 103, "button", BS_GROUPBOX |  
   WS_CHILD | WS_VISIBLE, 10, 5, 196, 49  
CONTROL "0.2", IDD_020, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE | WS_GROUP |  
   WS_TABSTOP, 24, 18, 37, 12  
CONTROL "0.4", IDD_040, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 24, 36, 37, 12  
CONTROL "0.8", IDD_080, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 69, 18, 37, 12  
CONTROL "1.0", IDD_100, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 69, 36, 37, 12  
CONTROL "2.0", IDD_200, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 114, 18, 37, 12  
CONTROL "3.2", IDD_320, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 114, 36, 37, 12  
CONTROL "5.0", IDD_500, "BUTTON", BS_AUTORADIOBUTTON |  
   WS_CHILD | WS_VISIBLE, 159, 18, 37, 12
CONTROL "6.4", IDD_640, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 159, 36, 37, 12
CONTROL ",", 102, "BorShade", 2 | WS_CHILD | WS_VISIBLE, 0, 63, 215, 1
SCROLLBAR IDD_SPEEDSB, 21, 97, 128, 9, SBS_HORZ | WS_CHILD | WS_VISIBLE | WS_TABSTOP
LTEXT ",", IDD_LS, 17, 109, 16, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT ",", IDD_HS, 137, 109, 16, 8, WS_CHILD |
LTEXT "Initial Speed :", -1, 17, 82, 47, 8, WS_CHILD |
LTEXT ",", IDD_IS, 64, 82, 16, 8, WS_CHILD |
LTEXT "mph.", -1, 80, 82, 21, 8, WS_CHILD |
CONTROL "Initial Speed", 105, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 10, 70, 149, 50
CONTROL ",", 106, "BorShade", 3 | WS_CHILD | WS_VISIBLE, 169, 63, 2, 113
CONTROL ",", 107, "BorShade", 2 | WS_CHILD | WS_VISIBLE, 0, 129, 170, 3
CONTROL "Constant Speed", IDD_FIXED, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 15, 154, 67, 10
CONTROL "Variable Speed", IDD_VARIABLE, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 87, 154, 67, 10
CONTROL "Object Speed", 109, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 10, 139, 151, 30
CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 176, 92, 32, 20
CONTROL "CANCEL", IDCANCEL, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 176, 137, 32, 20
END

// This dialog box is used to set up preview distance and object speed.

FPDbox DIALOG 57, 17, 214, 176
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CLASS "BorDlg"
CAPTION "Constant Preview Distance"
BEGIN
SCROLLBAR IDD_PDSB, 27, 31, 163, 9, SBS_HORZ | WS_CHILD |
| WS_VISIBLE | WS_TABSTOP

CTEXT "0", -1, 22, 43, 16, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
CTEXT "90", -1, 180, 43, 16, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
LTEXT "Preview Distance:", -1, 84, 14, 62, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP
LTEXT ",", IDD_PD, 144, 14, 16, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
LTEXT "ft.", -1, 160, 14, 16, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
CONTROL "Preview Dist.: feet", 103, "button", |
BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 10, 5, 196, 49
CONTROL "", 102, "BorShade", 2 | WS_CHILD | WS_VISIBLE, 0, 63, 215, 1
SCROLLBAR IDD_SPEEDSB, 21, 97, 128, 9, SBS_HORZ |
WS_CHILD | WS_VISIBLE | WS_TABSTOP
LTEXT "", IDD_LS, 17, 109, 16, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
LTEXT "", IDD_HS, 137, 109, 16, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP
LTEXT "Initial Speed:", -1, 17, 82, 47, 8, WS_CHILD |
WS_VISIBLE | WS_GROUP
LTEXT "", IDD_IS, 64, 82, 16, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
LTEXT "mph", -1, 80, 82, 21, 8, WS_CHILD | WS_VISIBLE |
WS_GROUP
CONTROL "Initial Speed", 105, "button", BS_GROUPBOX |
WS_CHILD | WS_VISIBLE, 10, 70, 149, 50
CONTROL "", 106, "BorShade", 3 | WS_CHILD | WS_VISIBLE, 169, 63, 2, 113
CONTROL "", 107, "BorShade", 2 | WS_CHILD | WS_VISIBLE, 0, 129, 170, 3
CONTROL "Constant Speed", IDD_FIXED, "BorRadio", |
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE |
WS_GROUP | WS_TABSTOP, 15, 154, 67, 10
CONTROL "Variable Speed", IDD_VARIABLE, "BorRadio", |
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 87, 154, 67, 10
CONTROL "Object Speed", 109, "button", BS_GROUPBOX |
WS_CHILD | WS_VISIBLE, 10, 139, 151, 30
CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 176, 92, 32, 20
CONTROL "CANCEL", IDCANCEL, "BorBtn", BS_PUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 176, 137, 32, 20
END
Appendix D

Resource Definition File (TRACER.RC) of TRACER

with Detailed Description
```c
#include <windows.h>
#include "tracer.h"

// The standard structure of resources in .RC file is
// NAME TYPE DATA

TRACERICON ICON "tracer.ico"
CABITMAP BITMAP "car.bmp"
CIBITMAP BITMAP "cir.bmp"
CRBITMAP BITMAP "cr.bmp"

// Name of main menu

MainMenu MENU
{
    POPUP "&File" // Popup or Top-Level Menu Item
        // Those menu ids (IDM_) are linked to tracer.h.
        // to tracer.h.
    {
        MENUITEM "Open .&dat File...", IDM_OPENDAT
        MENUITEM SEPARATOR
        MENUITEM "Open .&inf File...", IDM_OPENINF
        MENUITEM SEPARATOR
        MENUITEM "Open .&rst File...", IDM_OPENRST
        MENUITEM SEPARATOR
        MENUITEM "&Print", IDM_PRINT
        MENUITEM SEPARATOR
        MENUITEM SEPARATOR
        MENUITEM "E&xit", IDM_EXIT
    }
    POPUP "&Window"
    {
        MENUITEM "&Cascade\tShift+F5", IDM_CASCADE
        MENUITEM "&Tile\tShift+F4", IDM_TILE
        MENUITEM "Arrange &Icons", IDM_ARRANGE
        MENUITEM "C&lose All", IDM_CLOSEALL
    }
    POPUP "\a&About" // "\a" will result in the menu item being
    // put in the left side of the menu bar.
    {
        MENUITEM "A&bout Tracer...", IDM_ABOUT
    }
}

// Accelerator key table.
```
It is used to define accelerator keys or hot keys.

MenuAccel ACCELERATORS
{
    VK_F5, IDM_CASCADE, VIRTKEY, SHIFT
    VK_F4, IDM_TILE, VIRTKEY, SHIFT
}

Dialog box is used to exchange information with users. Those child window control ids (IDD...) are linked to tracer.h.

This dialog box is used to input file name.

Dial2 DIALOG 88, 49, 146, 108
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CLASS "BorDlg" // Borland style dialog box
CAPTION "Open File..."
BEGIN
    CONTROL ",", IDD_FILENAME, "static", SS_LEFT | WS_CHILD |
             WS_VISIBLE, 50, 9, 92, 8
    LTEXT "File Name:"", -1, 7, 9, 37, 8, WS_CHILD |
             WS_VISIBLE |
    CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD |
             WS_VISIBLE | WS_TABSTOP, 108, 49, 32, 20
    CONTROL "Cancel", IDCANCEL, "BorBtn", BS_PUSHBUTTON |
             WS_CHILD | WS_VISIBLE | WS_TABSTOP, 108, 80, 32, 20
    LTEXT "Path:"", -1, 7, 24, 20, 8, WS_CHILD | WS_VISIBLE |
             WS_GROUP
    CONTROL ",", IDD_PATH, "static", SS_LEFT | WS_CHILD, 33, 24, 109, 8
    CONTROL "Files...", -1, "button", BS_GROUPBOX |
             WS_CHILD |
    CONTROL ",", IDD_DIRECTOR, "LISTBOX", LBS_STANDARD |
             WS_CHILD | WS_VISIBLE | WS_TABSTOP |
    WS_VSCROLL, 12, 53, 81, 45
    CONTROL ",", 811, "BorShade", 3 | WS_CHILD | WS_VISIBLE, 104, 37, 42, 73
    CONTROL ",", 812, "BorShade", 2 | WS_CHILD | WS_VISIBLE, 104, 37, 43, 72
END

This dialog box is used to select display mode.

Dial3 DIALOG 80, 43, 130, 41
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CLASS "Bordlg"
CAPTION "Display Mode"
BEGIN
  CONTROL "Text Display", 910, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 10, 7, 67, 10
  CONTROL "Graphic Display", 920, "BorRadio", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 10, 23, 67, 10
  CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 92, 11, 32, 20
END

This dialog box is used to provide the information of TRACER

Aboutbox DIALOG 78, 44, 142, 92
STYLE WS_DLFRAME | WS_POPUP | WS_CAPTION | WS_BORDER
CLASS "BorDlg"
CAPTION "About Tracer..."
BEGIN
  CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 99, 110, 32, 20
  RTEXT "Ohio University", -1, 165, 110, 56, 8, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
  ICON "ThesisICON", -1, 103, 41, 16, 16, WS_CHILD | WS_VISIBLE
  LTEXT "Industrial and Systems Engineering", -1, 101, 99, 120, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
  RTEXT "June/1994", -1, 185, 120, 36, 7, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP
  CONTROL "SIMULATOR 1.0", -1, "STATIC", SS_CENTER | SS_NOPREFIX | WS_CHILD | WS_VISIBLE | WS_GROUP, 87, 10, 59, 8
  CONTROL ",", 102, "BorShade", 32769 | WS_CHILD | WS_VISIBLE, 76, 7, 80, 13
  CONTROL ",", 104, "BorShade", 32770 | WS_CHILD | WS_VISIBLE, 2, 23, 221, 113
  CONTROL ",", 103, "BorShade", 32769 | WS_CHILD | WS_VISIBLE, 101, 39, 20, 20
  CTEXT "Ergonomics Laboratory", -1, 100, 90, 109, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
  LTEXT "Thesis Advisor ":", -1, 130, 28, 54, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
  LTEXT "Dr. Helmut T. Zwahlen", -1, 130, 38, 91, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
// If the user wants to cancel the printing job, he can inform Windows by the dialog box.

AbortDlg DIALOG 103, 53, 110, 53
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CLASS "BorDlg"
CAPTION "Abort Printing"
BEGIN
  CONTROL "CANCEL", IDCANCEL, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 43, 27, 32, 20
  CTEXT "Send file to printer spooler! ", -1, 12, 11, 90, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
END

// This dialog box is used to choose data to display.

diall DIALOG 80, 47, 142, 136
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CLASS "BorDlg"
CAPTION "Display"
BEGIN

LTEXT "Which section do you want to look at?", -1, 8, 7, 130, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
CONTROL "1st, 0-1 min.", IDD_1SEG, "BUTTON",
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 20, 40, 55, 12
CONTROL "2nd, 1-2 min.", IDD_2SEG, "BUTTON",
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 20, 54, 55, 12
CONTROL "3rd, 2-3 min.", IDD_3SEG, "BUTTON",
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 20, 68, 55, 12
CONTROL "4th, 3-4 min.", IDD_4SEG, "BUTTON",
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 20, 82, 55, 12
CONTROL "5th, 4-5 min.", IDD_5SEG, "BUTTON",
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 20, 96, 55, 12
CONTROL "All Sections", IDD_ALLSEG, "BUTTON",
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 20, 110, 55, 12
CONTROL "Sections", 720, "button", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 13, 23, 70, 104
CONTROL "OK", IDOK, "BorBtn", BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 97, 77, 32, 20
CONTROL "CANCEL", IDCANCEL, "BorBtn", BS_PUSHBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 98, 107, 32, 20
LTEXT "Experiment ", -1, 93, 28, 41, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "Time ":", -1, 93, 41, 29, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
CTEXT ":", IDD_EXPTIME, 92, 56, 12, 8, WS_CHILD | WS_VISIBLE | WS_GROUP
LTEXT "min.", -1, 104, 56, 15, 8, WS_CHILD | WS_VISIBLE | WS_GROUP

END
Appendix E

User Manual
How to Use this Manual

Chapter 1: Installation and Introduction

Use this chapter to install hardware equipment and "SIMULATOR" and TRACER onto your computer system. This chapter also provides a brief introduction to the package.

Chapter 2: Using SIMULATOR

Use this chapter to learn how to perform a preview tracking experiment on SIMULATOR.

Chapter 3: Using TRACER

Use this chapter to learn how to use TRACER to analyze the experimental data that was generated by SIMULATOR.

Chapter 4: Troubleshooting

Use this chapter to learn how to solve some general problems that you might encounter during using of SIMULATOR and TRACER.
Chapter 1

Installation and Introduction

I. Installation

Before you start to install the package, you must make sure that the following items are available.

1. An Intel 80386 or 80486 CPU based personal computer with a VGA or better monitor.
2. A two-port game card compatible for use with the MAXX II YOKE and the MAXX-PEDALS.
3. A MAXX II YOKE and a MAXX-PEDALS.
4. The software installation diskette of this package.
   (Only one diskette)

1.1 Game Card Installation

1. Turn off the computer and remove the computer cover.
2. Find an unused extension slot (8 bits or 16 bits) and remove the blank bracket.
3. Hold the top edge of the game card and press it into the slot.
4. Fasten the card's bracket.
5. Re-assemble the computer cover and screws.
6. Turn on the computer.
7. Install the game card driver.
8. If you use the KRAFT game card, make sure that the
following commands have been added into your AUTOEXEC.BAT which is found in the boot directory of your computer.

```
PATH = C:\CSOFT......... (the directory name)
KCARD 4
KEYCON
```

If you use other kinds of game card, please refer to the user's guide offered from the manufacturer of your game card.

1.2 MAXX II YOKE and MAXX PEDALS Installation

1. Turn off your computer.
2. Plug the MAXX PEDALS into Port A of your game card.
3. Then, plug the MAXX II YOKE to the Y-connector of the MAXX PEDALS.
4. Restart your computer.

1.3 Software Installation

1. Insert the installation diskette in your 3.5 inch floppy disk drive.
2. Change your directory to that drive. For example, a:
or b:
3. Type

```
install <source-drive> <destination-drive>
```

followed by the <Enter> Key.
<source-drive>: Floppy drive containing the installation disk, like a:, b:.
<destination-drive>: Hard disk on which you want to install the package. Only three hard drives, c:, d:, and e:, are supported.

For example, **install a: d:**
then press the <Enter> key.
This would install the package from drive a: to drive d:.

II. Introduction

The package is designed as a true Windows application to execute the preview tracking experiment. There are two programs: SIMULATOR and TRACER.

The prime duty of SIMULATOR is to execute the preview tracking experiment and to save the experimental data on a mass storage device. If you want to start a preview tracking experiment, you have to double click the SIMULATOR icon in the Program Manager Window.

The main job of TRACER is to analyze the experimental data that was generated by SIMULATOR. TRACER generates a statistical analysis report. If you would like to review the experimental data or check the experimental results, you
must double click the TRACER icon in the Program Manager Window.
Chapter 2

Using SIMULATOR

I. How to start the program

1. Double click the SIMULATOR's icon in the "TRACKING" group to start the program.

2. An initial window showing the title and the copyright information of SIMULATOR will be presented on the screen.

Figure 2a illustrates the initial window of SIMULATOR.
Figure 2a The initial window of SIMULATOR
II. Menu Items

There are two pop-up menu items assigned to the menu bar of SIMULATOR. The first pop-up menu item is Options. It has two sub menu items, New Subject... and Exit. The second pop-up menu item is About and has only one sub menu item, About SIMULATOR...

Figures 2b & 2c illustrate the menu items of SIMULATOR.
Figure 2b The main menu item -- Options
Figure 2c. The main menu item -- About
III How to start a new experiment

1.1 Use your mouse to point to the sub menu item of **New Subject**..., a small dialog box will be opened on the screen, allowing you to enter the subject's ID. This is illustrated in figure 2d.

1.2 After entering the subject's ID, click the **OK** button to enter the next dialog box.
Figure 2d The dialog box to enter the subject's ID
2.1 Then, the dialog box of "Variables Setup 1" is opened. This is illustrated in figure 2e. You can select the desired preview type by clicking the corresponding radio button.
Figure 2e The dialog box of "Variable Setup 1"
2.2.1 If you select "Fixed Time", a new dialog box, "Constant Preview time," will be opened. You can set the values of the preview time, the initial speed, and the object speed which may be either constant or variable. Figure 2f illustrates the dialog box "Constant Preview Time".

2.2.2 Then, click OK to return to "Variables Setup 1".

If you select Variable Speed in the group box of the Object Speed, you may use the pedals to adjust the object speed. Otherwise, the pedals are inactive.
Figure 2f The dialog box of "Constant Preview Time"
2.3.1 If you choose "**Fixed Dist**", a new dialog box "**Constant Preview Distance**" will be opened. You can set the values of the preview distance, the initial speed, and the object speed which may be either constant or variable. Figure 2g illustrates the dialog box "**Constant Preview Distance**".

2.3.2 Then, click **OK** to return to **Variables Setup 1**.

If you select the variable speed in the group box of the Object Speed, you may use the pedals to adjust the object speed. Otherwise, the pedals are not active in this mode of operation.
Figure 2g The dialog box of "Constant Preview Distance"
2.4 Then, you can adjust the values of the initial seed and the road width in "Variables Setup 1".

2.5 Click **OK** to enter the next dialog box, or click **Cancel** to return to the main window.
3.1.1 Then, "Variables Setup 2" dialog box will be presented on the screen. You can set the values of Relative Curve Rate, Time Lag, Input Gain, Object Shape, and Center Line in this dialog box. Figure 2h illustrates this.

3.1.2 Click **OK** to enter the next dialog box, or click **Cancel** to return to the main window.
Figure 2h The dialog box of "Variable Setup 2"
4.1.1 Then, the "Variables Setup 3" dialog box will be presented on the screen. You can set the values of Experiment Time, Subject Adaption Time, and the curve direction of the First Curve in this dialog box. Subject Adaption Time is the beginning period of tracking while data will not be recorded and used in analysis. Figure 2i illustrates this dialog box.

4.1.2 Click OK to start the experiment or click Cancel to return to the main window.
Figure 2i The dialog box of "Variable Setup 3"
5.1 Then, the experiment is started. The subject can control the lateral position of the object by MAXX YOKE and MAXX PEDALS. Figure 2j illustrates the screen of the experiment.

Except the sub-window "WindShield" which is used to display the road and car routes, there are two more sub-windows being presented on the screen. The "SpeedBar" is used to notify the subject of how fast his controlled object is. The red bar shows subjects the current object speed. The "BrakeBlock", which is located in the right-lower corner, is used to notify the subject whether the brake is applied or not. If the brake is applied, the color of "BrakeBlock" will be changed from blue to red.
Figure 2j The 3 sub-windows of SIMULATOR, "WindShield", "SpeedBar", and "BrakeBlock".
6.1 After finishing the experiment, a new dialog box "Data Save..." appears on the screen. If you want to store the experimental data on a diskette, you may choose OK. Otherwise, the experimental data will be discarded. Figure 2k illustrates the dialog box "Data Save...".
Figure 2k The dialog box "Data Save..."
7.1.1 If you click OK in the dialog box "Data
Save...", the dialog box "Save..." will appear on
the screen. You may enter your file name and
select the directory to store data. Figure 21
illustrates the dialog box "Save...".

7.1.2 Click OK to save the data or click Cancel to
discard it. The program will return to the main
window.
Figure 21 The dialog box “Save...”
IV. About the SIMULATOR program

1.1 If you want to know more about the program, you may activate the sub menu item About SIMULATOR...

A dialog box "About SIMULATOR" will appear on the screen. The box contains information about the version of SIMULATOR, the SIMULATOR icon, the thesis project title, the thesis advisor's name, the developer's name, the copyrights, and the completion date of this program. Figure 2m illustrates the dialog box "About SIMULATOR".
Figure 2m The dialog box "About SIMULATOR"
Chapter 3

Using TRACER

I. How to start the program

1. Double click TRACER icon in the "TRACKING" group to start the program.

2. An initial window of TRACER will be presented on the screen and lasted for 6 seconds. After that, the window will be closed automatically. This initial window can offer copyright information of the program to the users.

Figure 3a illustrates the initial window of TRACER.
Figure 3a The initial window of TRACER
II. Menu Items

There are three pop-up menu items assigned the menu bar of TRACER. The first Pop-up menu item is **Files**. It contains five sub menu items, **Open .dat file**, **Open .inf file**, **Open .rst file**, **Print**, and **Exit**. An extra space is prepared behind **Exit** to show the windows have been opened, like **About TRACER** in figure 3b. The second pop-up menu item is **Window** and it has four sub menu items, **Cascade**, **Tile**, **Arrange Icon**, and **Close all**. The last pop-up menu item **About** has only one sub menu item, **About TRACER**.

Figure 3b, 3c & 3d illustrate the menu items of TRACER.
Figure 3b The menu item "Files"
Figure 3c The menu item “Window”
Figure 3d The menu item "About"
III. How to open a .dat file

1.1 Click the sub menu item "Open .dat file...".

1.2 A dialog box "Open File..." is presented on the screen, allowing you to picking the desired data file.

1.3 Double click at the file name that you want. (You can click at the file name to highlight it. In order to avoid unwanted operation, only double click can pick up the file name.)

1.4 Click OK to acknowledge the file name, or click Cancel to return to the main window.

Figure 3e illustrates the dialog box "Open File...".
Figure 3e The dialog box "Open File..."
2.1 If you select OK in "Open File...", a new dialog box "Display Mode" is presented on the screen allowing you determining the display mode which can be either text display or graphical display.

2.2 Use the radio buttons to set the display mode of your choice, then click OK.

Figure 3f illustrates the dialog box "Display Mode".
Figure 3f The dialog box "Display Mode"
3.1 A new dialog box "Display" will appear on the screen. You may select one or more data sections to analyze. Figure 3g illustrates the dialog box "Display".

3.2 Click **OK** to open the data file or click **Cancel** to return to the main window.
Figure 3g The dialog box "Display"
4.1.1 If you select the "Text Display" in "Display Mode", you will see a window as shown in figure 3h. You may scroll the window to look up the entire experimental data by using the scroll bar.
<table>
<thead>
<tr>
<th>Time</th>
<th>Left Road</th>
<th>Car Pos.</th>
<th>Car Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>069.00</td>
<td>088.00</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>069.00</td>
<td>088.50</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>063.00</td>
<td>089.25</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>068.75</td>
<td>089.75</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>067.50</td>
<td>090.25</td>
<td>030</td>
</tr>
<tr>
<td>0005</td>
<td>065.75</td>
<td>090.50</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>064.00</td>
<td>090.50</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>062.25</td>
<td>089.25</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>060.50</td>
<td>087.25</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>058.75</td>
<td>084.75</td>
<td>030</td>
</tr>
<tr>
<td>0010</td>
<td>057.00</td>
<td>082.25</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>055.25</td>
<td>079.75</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>053.50</td>
<td>077.25</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>051.75</td>
<td>075.00</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>050.00</td>
<td>072.75</td>
<td>030</td>
</tr>
<tr>
<td>0015</td>
<td>048.25</td>
<td>070.50</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>046.50</td>
<td>068.25</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>044.75</td>
<td>066.00</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>043.00</td>
<td>063.50</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>041.25</td>
<td>061.25</td>
<td>030</td>
</tr>
<tr>
<td>0020</td>
<td>039.50</td>
<td>059.75</td>
<td>030</td>
</tr>
<tr>
<td></td>
<td>037.75</td>
<td>059.75</td>
<td>030</td>
</tr>
</tbody>
</table>

**Figure 3h** The window "Text Display"
4.2.1 If you select the "Graphic Display" in "Display Mode", you will see a window as illustrated in figure 3i. You may scroll the window to look up the entire experimental data by using the scroll bar.
Figure 3i The window "Graphic Display"
IV. How to open a .inf file

1.1 Click the sub menu item "Open .inf file...".

1.2 A dialog box "Open File..." is presented on the screen, allow you to pick the desired data file.

1.3 Double click at the file name of your choice.

1.4 Click OK to acknowledge the file name, or click **Cancel** to return to the main window.

Figure 3j illustrates the dialog box "Open File...".
Figure 3j The dialog box "Open File..."
2.1 Figure 3k illustrates the window of the selected .inf file.
Figure 3k The window "Information Display"
V. How to open a .rst file

1.1 Point the sub menu item "Open .rst file...".

1.2 A dialog box "Open File..." is presented on the screen, allowing you to pick the desired data file.

1.3 Double click the file name of your choice.

1.4 Click **OK** to acknowledge the file name, or click **Cancel** to return to the main window.

Figure 31 illustrates the dialog box "Open File...".
Figure 31 The dialog box "Open File..."
2.1 Figure 3m illustrates the window of the selected .rst file.
Figure 3m The window "Result Display"

Car (C) Position With Respect To Right Road Side
Mean = - 15.547 ft. (- means left side.)
Std. Dev. = 6.6653 ft.
Average Speed = 030.000 mph.
Off Road Dist. = 0167 ft.
Off Road Time = 3.8 sec.
VI. How to print the content of the window

1.1 Click the sub menu item, **Print**, then the content of the current active window is transferred to the printer spool.

1.2 A dialog box "**Aborting Printing**" is presented on the screen. You may click **Cancel** to stop the printing job. Figure 3n illustrates the dialog box of "**Abort Printing**".
Figure 3n. The dialog box "Abort Printing"
VII. Windows management

1.1 Click the sub menu item **Cascade**, to arrange the opened windows as shown in figure 30.

1.2 Click the sub menu item **Tile**, to arrange the opened windows as shown in figure 3p.

1.3 Click the sub menu item **Arrange Icon** to rearrange the positions of the window icons.

1.4 Click the sub menu item **Close All** to close all opened and non-active child windows.
Figure 30 Cascade the opened windows
VIII. About the TRACER program

1.1 If you want to know more about the program, you may click the sub menu item About TRACER.... A dialog box "About TRACER" will appear on the screen, containing information about the version of TRACER, the TRACER icon, the thesis project title, the thesis advisor's name, the developer's name, the copyrights, and the complete date of the program. Figure 3q illustrates this dialog box.
Chapter 4

Troubleshooting

I. Solutions to problems encountered when using SIMULATOR

1.1 Problem: The generated road is always straight, no matter how large the relative curve rate is selected.

Reason: The seven .INI files are not in SIMULATOR's own working directory.

Possible conditions and Solutions:

(1) Copy *.INI files into SIMULATOR's working directory.

(2) In the Program Manager window,
   (I) click File on menu bar,
   (II) click Properties... on sub menu list, and
   (III) check the Working Directory: of SIMULATOR.

1.2 Problem: The object, car or circle, is not controllable by the subject.

Reason: The input values from yoke and pedals are not received by the system.

Possible Conditions and Solution:

(1) Recheck your game card driver.
(2) Replug the connectors of Yoke and pedals and make sure that they are plugged on the right positions.

(3) Make sure that the interrupt 15h does not occupied by the other TSRs (Terminate-and-Stay-Resident programs.) You can use MSD.EXE, a program come with MS-DOS 5.0 and later, as well as MS-Windows 3.1x to check interrupts used or not.

1.3 Problem: The experimental data is lost or not saved in assigned diskette and directory.

Reason: file name was not entered or entered in an invalid format.

Possible Conditions and Solutions:

(1) Repeat the experiment. It is impossible to find the lost data.

II. The general troubles and solutions in using TRACER

1.1 Problem: The generated window contains no text, graphic, or analysis report.

Reason: You have exhausted all the available memory or system resources.
Possible conditions and Solutions:

(1) Check the Windows function "About Program Manager..." to inspect how much memory and system resources are available.

(1) Close all windows, then restart TRACER.
Appendix F

Data Structure of .DAT, .INF, and .RST Files
1. Data structure of .DAT file

<table>
<thead>
<tr>
<th>Digit</th>
<th>Content</th>
<th>Type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>Experiment Time</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>4-6</td>
<td>Subject Adaptation Time</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>7-9</td>
<td>Road Width (ft.)</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>10-12</td>
<td>Object Width (pixel)</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>(9x+13)-(9x+15)</td>
<td>X-coordinates of Left Road</td>
<td>INT</td>
<td>x=0..499</td>
</tr>
<tr>
<td>(9x+16)-(9x+18)</td>
<td>X-coordinates of Object</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>(9x+19)-(9x+21)</td>
<td>Object Speed</td>
<td>INT</td>
<td></td>
</tr>
</tbody>
</table>

2. Data structure of .INF file

<table>
<thead>
<tr>
<th>Digit</th>
<th>Content</th>
<th>Type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>Preview Distance</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>4-6</td>
<td>Road width (ft.)</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>7-9</td>
<td>Initial Speed</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>10-12</td>
<td>Object Speed</td>
<td>INT</td>
<td>750:Con751:Var.</td>
</tr>
<tr>
<td>13-21</td>
<td>Initial Seed</td>
<td>LONG INT</td>
<td></td>
</tr>
<tr>
<td>22-24</td>
<td>Relative Curve Rate</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>25-30</td>
<td>Time Lag (msec.)</td>
<td>LONG INT</td>
<td></td>
</tr>
<tr>
<td>31-33</td>
<td>Input Gain</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>34-36</td>
<td>Object Shape</td>
<td>INT</td>
<td>907:Car908:Cir.</td>
</tr>
<tr>
<td>37-39</td>
<td>Central Line</td>
<td>INT</td>
<td>909:Yes 910:No</td>
</tr>
<tr>
<td>40-42</td>
<td>Experiment Time</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>43-45</td>
<td>Subject Adaptation Time</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>46-48</td>
<td>Curve Direction</td>
<td>INT</td>
<td>870:R 871:L</td>
</tr>
<tr>
<td>49-51</td>
<td>Preview Type</td>
<td>INT</td>
<td>1:Dist. 2:Time</td>
</tr>
<tr>
<td>52-54</td>
<td>Preview Time</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>55-74</td>
<td>Subject ID</td>
<td>CHAR (ASCII)</td>
<td></td>
</tr>
</tbody>
</table>
3. Data structure of .RST file

<table>
<thead>
<tr>
<th>Digit</th>
<th>Content</th>
<th>Type</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-6</td>
<td>No. of Car in Position 1</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>7-12</td>
<td>No. of Car in Position 2</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>13-18</td>
<td>No. of Car in Position 3</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>19-24</td>
<td>No. of Car in Position 4</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>25-30</td>
<td>No. of Car in Position 5</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>31-36</td>
<td>No. of Car in Position 6</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>37-42</td>
<td>No. of Car in Position 7</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>43-45</td>
<td>Not Used</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46-48</td>
<td>Object Shape</td>
<td>INT</td>
<td>907:Car 908:Cir.</td>
</tr>
<tr>
<td>49-54</td>
<td>Mean Position x 100</td>
<td>LONG</td>
<td></td>
</tr>
<tr>
<td>55-57</td>
<td>Positive or Negative Mean</td>
<td>INT</td>
<td>0:Pos. 1: Neg.</td>
</tr>
<tr>
<td>58-66</td>
<td>Standard Deviation x 1000</td>
<td>LONG</td>
<td></td>
</tr>
<tr>
<td>67-72</td>
<td>Average Speed x 100</td>
<td>LONG</td>
<td></td>
</tr>
<tr>
<td>73-75</td>
<td>Experiment Time</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>76-78</td>
<td>Subject Adaptation Time</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>79-84</td>
<td>Off Road Distance (pixel)</td>
<td>LONG</td>
<td></td>
</tr>
<tr>
<td>85-90</td>
<td>Off Road Time (msec.)</td>
<td>LONG</td>
<td></td>
</tr>
<tr>
<td>91-93</td>
<td>Central Line</td>
<td>INT</td>
<td>909:Yes 910:No</td>
</tr>
</tbody>
</table>