1201

DESIGN AND IMPLEMENTATION OF AN OBJECT-ORIENTED GRAPHICAL USER INTERFACE FOR A RAILWAY RESERVATION SYSTEM

Dissertation submitted to the Jawaharlal Nehru University in partial fulfilment of the requirements for the award of the Degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE & TECHNOLOGY

by

B.V.RAVEENDRA

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI 110067
INDIA

JANUARY 1995

CERTIFICATE

This is to certify that the dissertation entitled **"DESIGN AND IMPLEMENTATION OF AN OBJECT-ORIENTED GRAPHICAL USER INTERFACE FOR A RAILWAY RESERVATION SYSTEM"** , being submitted by **B.V.RAVEENDRA** to Jawaharlal Nehru University, New Delhi in partial fulfilment of the requirements for the award of the degree of **Master of Technology** in Computer Science and Technology is a record of the original work done by him under the supervision of **PROF. K.K.NAMBIAR**, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi during the year 1994, monsoon semester.

The results reported in this dissertation have not been submitted in part or full to any other university or Institution for the award of any degree or diploma.

[illegible] 16-1-95

Prof. K.K. Bharadwaj, Dean
School of Computer &
Systems Sciences,
Jawaharlal Nehru University
New Delhi - (INDIA)
Pin 110067.

[illegible]

Prof. K.K. Nambiar,
School of Computer &
Systems Sciences,
J.N.U., New Delhi.

ACKNOWLEDGEMENTS

I wish to express my sincere thanks and gratitude to Prof. K.K. Nambiar for his invaluable guidance and the enthusiasm with which he helped me during the course of this project work.

I would like to thank Prof. K.K. Bharadwaj, Dean of School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi, for making the various facilities available.

I am grateful to other members of teaching and non-teaching staff who helped in all measures to make this project a success.

[illegible]

(B.V. RAVEENDRA)

"GURŌSTU MOUNAM VYĀKHYĀNAM |

SISHYĀSTU CHINNA SAMṠAYĀH ||"

Dedicated to all masters of all times, who stood for imparting invaluable knowledge and guided us towards light and bliss.

CONTENTS

CHAPTER 1

ABSTRACT

A comparative study between Windows programming and Object Windows programming using Object Windows Library (OWL) is carried out, highlighting the benefits of using OWL.

A complete database and a user interface are designed and implemented for the prototype of a Railway Reservation System; using the Borland C++ for Windows (BCW) and the Borland's Resource Workshop (BRW). The system can reserve a ticket, cancel a ticket, display any particular ticket's details, can answer all the enquiries for trains and fares details. The system is also equipped with a full help facility.

CHAPTER 2

INTRODUCTION

This project is an attempt towards the design and implementation of a prototype for a Railway Reservation System with MSWindows 3.1 Graphical User Interface(GUI) using Borland C++ for Windows with OWL.

Object Orientation makes writing programs simpler, easier and effective.The early development environments for Windows did little to help the programmer deal with the daunting task of writing a Windows program.Even a simple windows program requires much code to be written.The user is forced to sift through over 600 API functions to find a suitable one that achieves his task.Borland C++ with Object Windows takes some of the pain out of Windows programming by providing predefined object classes that handle many of the details of Windows programming.An Object Windows program consists of lesser code when compared to a Windows program without Object Orientation.New features can be added to the program by just adding the appropriate object to it.Because Object Windows is an object oriented library,the programmer can inherit the features needed from the library and overload the routines that are needed to change.

The state-of-the-art GUI's have taken the computer industry by storm and liberated the user from all of those arcane sequences of key strokes to accomplish his task.

With the advent of more "user friendly" Windows, users began to work on it playfully, because it is easy to use "visual interface" with its asthetic appeal. Today the user can get all his work done just at the click of a mouse button. In addition to the textual data, the directories, applications, peripherals, accessories are represented by Icons. These icons depict to the user the qualities and characteristics of those concepts or items of which these icons are tokens. So, user can easily notice the item he wants instead of searching through directories etc. Moreover, by just moving and clicking on icons, user gets his work done in certain applications. For example, in a data communication application, by simply selecting the icon representing a baud rate and the icon representing the type of channel; the user can change the baud rate to the new specification as well as the communicaton channel being used, say for example, from coaxial cable to fibre optic cable. What a fantastic flexibility ! Thanks to Windows User Interface.

As Windows 3.1 includes the Multi-media extensions, new types of applications can be written. These burgeoning relatively new technologies will bring computers to people who have never used a computer before. With special versions of Windows, imagine yourself being able to program your VCR, CD player, TV, microwave oven and so on, in the days to come.

Wayback in mid-1970's, the machines such as the Alto and the Star and environemnts such as Smalltalk supported the visual interface. Subsequently with the bloom of Apple's Macintosh GUI's have flooded the computer industry. Present day GUI's available are :

1. for IBM - compatibles		
(i) running MS-DOS	------->	Windows
(ii) running OS/2	------->	Presentation Manager
2. for the Commodore Amiga	------->	Intuition
3. for the Atari	------->	GEM
4. for machines running UNIX	------->	X-Windows
5. for Sun Micro systems workstation	------->	NeWs
6. for the NeXT	------->	NextStep

Microsoft Windows version 3.1 was released in April 1992. The most significant new feature in Windows 3.1 is the True Type font technology developed by Apple Computer, Inc., and Microsoft . It gives scalable outline fonts to Windows. Windows 3.1 also supports Multimedia(sound & music), Object Linking and Embedding(OLE), and common dialogboxes. Windows 3.1 runs only in protected mode and requires an 80286 or 80386 processor with atleast 2MB of memory.

Windows 3.1 contains a new header file ,WINDOWSX.H . This file contains a number of macros to make writing Windows applications much easier and faster.Mostly,these macros are of great help in the casting operations; instead of yourself sprinkling the casts throughout the code.

WHAT IS WINDOWS ?

Windows is a graphics-based multitasking windowing environment that allows programs written specifically for Windows to have a consistent appearence and command structure.

The various built-in routines of the Windows package allow the easy implementation of pull-down menus, scroll bars, dialog boxes, icons, and many other features of a user-friendly graphical interface.Usage of the extensive graphics programming language allows a program to easily format and output text in a variety of fonts and pitches.

The video display, keyboard, mouse, printer, serialport,and system timers-all are default by windows in a device independent manner. This allows the same programme to run identically on a variety of hardware configurations.

The power of Windows lies in its three main capabilities:

-a graphics oriented user interface

-a multitasking capability

-and hardware independence.

A WINDOW DEFINED :

"To the user, a Windows window is a rectangular portion of the display with a program independent consistent appearence, that is the visual interface between the user and the program generating the window."

"To an application, the window is a rectangular area of the screen that is under the control of the program."

Each window possesses a Titlebar, a ControlBox, a MinimizeBox, a MaximizeBox, a window frame in minimum,and may also include scrollbars and other features. The management of an application window in effect is the cohesive effort between the application and Windows. The Windows maintains the standard display of an application window at a proper position and acts as a mediator between the user/application and the window, passing on the user input through messages to window. The window's client area can be completely organized by the user.

This whole activity is achieved through the Application Programming Interface(API) provided by the GUI. The API facilitates to create screen objects, to draw screen objects and to monitor mouse activations.

WHY WINDOWS GUI ?

Windows is a standardised GUI. The consistent user interface employs pictures to depict drives, files, subdirectiries and many other operating system commands and actions.Each program is identified by its caption bar.

One of the major advantages in using Windows GUI lies in the facility thatmany of the basic file manipulation functions are accessed through the program's menus by just a click of the mouse at the proper location.

Multitasking :

In the Windows multitasking environment the several application programs, or several instances of the same program are allowed to run concurrently.At any time the user is at ease to move the windows around on the screen, to switch between different programs,to change the window's size, and he can exchange information from window to window aswell. Windows is hence said to be a "desktop metaphor " for the display of multiple programs.Every program under Windows can be visualised as a RAM-resident popup.

Queued Input :

Under Windows, an application does not make explicit calls to get input from the keyboard or mouse. Instead it is the Windows that receives all inputs from mouse, keyboard, timer etc. in a 'System Queue'.Windows manages to redirect the input to the appropriate program by copying it from the System Queue into the program's queue.When the application program is ready to process its input, it directly reads from its own queue and dispatches a message to the correct window.

Information is disseminated in a multitasking environment through "messages".Messages can be generated by the user externally or by the program or by the Windows itself. A message is a notification that some event of interest has occured. The message in turn may or may not prompt any action.

<u>Device Independence</u> :

Hardware device independence is another colourful feather in the cap of Windows.Windows absolutely frees the programmer from having to cater for every possible variety of monitor, printer and input device available. In the Windows environment, each device driver need to be written only once.Windows includes a graphics programming language called the Graphics Device Interface(GDI) that allows the easy display of formatted text. Windows virtualizes display hardware.

Compilation Steps For a Windows Program :

1.The program's source code is compiled by the C/C++ compiler to produce an objective file.

2. The Linker takes both this Objective file as well as the module definition file and produces an unfinished .EXE file.

3. The Resource Compiler takes the resource file content, compiles it to form a resource data file with .RES extension.

4. This .RES file and the unfinished .EXE file, both are fed to the Resource Compiler to produce the final finished .EXE file of the program.

<u>Hardware and Software requirements to run Windows</u> :

The minimum hardware required is a 80286 based PC with 2MB of memory, a hard disk, and an EGA or VGA video screen. The software requirement is Borland C++ for Windows (BCW) along with Windows 3.1 package.

<u>STEPS IN CREATING A WINDOWS PROGRAM</u> :

Writing a Windows program can involve all or some of the following seven steps :

1. Write the Main program file with all the associated Windows functions in C or C++ language using the editor.
2. All the optional resources required such as Menu, DialogBoxes, ListBoxes, etc., are created using Borland's resource Workshop(BRW).
3. The corresponding resource scripts generated are added to the resource script file(.RC file) using "save project" option of BRW.

4. The Icon generation for the program, building the optional cursors & bitmaps also is achieved using BRW .
5. The custom module definitions are written and placed in a module definition file(.DEF file).
6. Make a project file with all the source codefile, .RC file, (.DEF) file, and the .H file if any.
7. Compile the project file using the "BuildAll" option of BCW compiler.

   Then finally run the program using the "Run" option.

The GUI designed uses the database created and responds to the requests of user. The functions that the system can carry out are :

1. Reserving a Railway Ticket.
2. Cancelling a Railway Ticket.
3. Response to Enquiry on vacancy.
4. Response to Enquiry on various Trains details
5. Response to Enquiry on fare from one station to another.
6. Display of a Reserved Ticket with all details, on production of its PNR no.
7. Listing of all the tickets Reserved.
8. Provision for the database administrator(DBA) to build the tree with all records of tickets reserved till the prior day intact and being ready to reserve next ticket.

Chapter3 introduces the basic concepts of Windows and explains how to write specific Windows applications.It also throws light on the memory management capabilities of Windows, knowledge of which aids the programmer to write efficient programs.Chapter4 focusses the design and implementation of the prototype of the GUI for the Railway Reservation System developed,with all details of the database design.Chapter5 tokens a sample session with the application program developed.chapter6 highlights the enhancements that can be made further to this project in future.

References used in the project are appended at the end of the thesis.

CHAPTER 3

ANATOMY OF A WINDOW

A standard Windows program will consist of some or all of the following, depending on which resources are being used :

1) The main programm file consisting of the code,

2) The Resource file where the resources being used are defined,

3) The header file, comprising all the declarations, and finally

4) The module definition file, which contains the definitions of heapsize, stacksize etc. required for the program.

Every Windows program <u>must</u> include the WINDOWS.H file. WINDOWS.H contains definitions for types WORD, HANDLE, HWND (handle of a window) etc etc. For example, the sample definitions are as below:

Standard Windows Data Types

Prefix	**Data Type Represented**
b	BOOL/(integer)
by	BYTE/(unsigred Character)
c	Character
dw	DWORD/(unsigned long)
fn	HANDLE/(unsigned integer)
i	integer

l	LONG/(long)
lp	long/(far) pointer
n	short integer
np	near/(short) pointer
p	pointer
s	string
sz	NULL/(o) terminated string
W	WORD/(unsigned integer)
X	short/(when used as the X coordinate)
y	short/(when used as the Y coordinate)

3.1 The WinMain ( ) Function:

The standard WinMain function format is as below:

```
int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpsz(mdLine, int nCmdShow)
```

Observe, this function is declared to be "int PASCAL". It is because the function returnes an integer value when it is finished. The word "PASCAL" directs the C compiler to use the function calling conventions of the Pascal language, rather then those of the C language. The basic reason for this is that the PASCAL function calling convention results in less computer code on compilation into machine instructions. The PASCAL calling convention puts the parameters passed by a function on the stack in the same order they are declared in the function's declaration. The C language convention does just the reverse. So, the compiler knows priorhand that the first argument is on the top of the stack with the PASCAL convention, while with the C convention, the Compiler is forced to search down the stack for the location of the first argument. But, notice that the PASCAL function calling convention doesn't allow functions to have variable number of arguments.

Thus Windows uses the PASCAL function calling convention to make programs smaller and faster.

Every Windows program must have WinMain( ) function. When a Windows application starts, it is the WinMain( ) function that begins execution. Similarly at the and of and of an application, WinMain( ) returns exit code to Windows.

The data type HANDLE indicates that handles are unsigned integers. Each window on the screen has a unique window handle. Handles are also used to uniquely identify each running application, each assigned memory block etc. The Windows kernel maintains tables for the convertion of these handles to physical memory addresses. A handle is basically a pointer to a pointer to a memory location. For memory management, if Windows moves the program or memory block in memory, the corresponding handle table is updated properly.

The program is abstract to all these changes. It can go ahead with its own defined handles, without any calamity.

WinMain( ) parameters:-

<u>hInstance</u> : These can be many applications running in Windows simultaneously. To distinguish among them, Windows creates and assigns these "hInstance" handles to each of them. Physically, hInstance is the memory handle to the application's default data segment.

<u>hPrevInstance</u>: There can be several instances of the same application program running in Windows at a time. To cope with these copies of the same application, as soon as another copy is started, hPrevInstance will contain the hInstance value for the last copy started. If only one instance of an application is running, hPrevInstance will be zero.

<u>lpszCmdLine</u>: This is a pointer to a character string containing the commandline arguments passed to the program, if any. If none is passed, it is NULL.

<u>nCmdShow</u>: This is the integer value to be passed to the Show Window( ) function.

3.2 Window Creation & Message processing:

The CreateWindow( ) function is used to create the application's window, having the window frame, captionbar, centerarea (called the "client area"). The 'style" of window, font for the letters on controls, the shade of the client area, the size & location of window etc. are determined by the CreateWindow( ) function. ShowWindow( ) function makes the window created to be visible on the screen.

3.2.1 Registering a window class:

For creating a new class of Windows, Register class() function is used. WNDCLASS structure is defined in WINDOWS.H. All the members of the WNDCLASS structure are filled first and then a pointer is passed to Register Class(). A standard window class structure declaration looks like below:

```
WNDCLASS         wndclass;
if (!h PrevInstance)
{
wndclass.style   = CS_HREDRAW | CS_VREDRAW;
wndclass.lpfnWndProc = WndProc;
wndclass.cbCls Extra = 0;
wndclass.cbWndExtra  = 0;
wndclass.hInstance = hInstance;
wndclass.hIcon  =LoadIcon (NULL, IDI_APPLICATION);
wndclass.hCursor    = Load Cursor (NULL, IDC_ARROW);
wndclass.hbrBackground  =  Getstockobject( WHITE_BRUSH);
wndclass.lpsz MenuName =NULL;
wndclass.lpsz ClassName = "myclass";
if (! Register class (& wndclass))
return (0);
}
```

If the same application is running more that once simultaneously, it is enough to register class only once. The style member of the WNDCLASS specifies that windows of this class should be redrawn if either the horizontal or vertical size changes. The cbClsExtra and cbWndEstra members of the WNDCLASS structure allow to allocate a memory block for each class or window of a class.

3.2.2 Message Loop:

Windows generates messages corresponding to the actions of user.

For example: The mouse cursor moved to X,Y on the screen (WM _ MOUSE MOVE)

The left mouse button was depressed

(WM _ LBUTTONDOWN)

A key was pressed

(WM _KEYDOWN).

Each message has a unique integer ID value. Windows applications cooperate with the Windows environment by continually checking whether there are any incoming messages. This is achieved through a small program loop called a "message loop". The message loop will use either the GetMessage() or PeekMessage() function to read messages sent to the program.

Within WinMain( ), a local variable ‘msg’ of type MSG is declared first. MSG data structure is defined in WINDOWS.H as

```
typedef struct tagMSG
{      HWND       hwnd;
WORD       message;/ *the message ID value */
WORD             wParam;/  *one WORD of data passed
                   with message */
LONG       lParam;/ *LONG data passed with message */
DWORD      time;
POINT      pt;
}MSG;
```

A message can be attributed to a block of memory, divided into the six different types of data defined in the MSG datastructure. The HWND and WORD data types occupy two bytes of memory each. The LONG, DWORD and POINT data-types all occupy four bytes. This results in a requirement of 18 bytes of memory per message when Windows sends a message to a running application. Windows fills in the data in a MSG datastructure and stores the data in a memory block that the application can read. The program calls either the **GetMessage()** or **PeekMessage()** function to read the message data. The first parameter passed to the GetMessage() function is a pointer to a message structure. GetMessage() copies the message data into this structure so that the program can use it.

When a program executes GetMessage(), the Windows environment checks whether there are any messages waiting for the program. If there are none, Windows doesn't allow the GetMessage() program loop to run. Instead, Windows retains control of the system and goes about doing other things until a message to the program is generated. Only if there is a message for the program, the GetMessage() function gets it and the subsequent actions get triggered, allowing the program to start operating. GetMessage() is one of the keys to Windows' ability to run several programs at the same time. The DefWindowProc() function carries out all of the default actions for a window. The outlining of the window, movement, and repainting is all handled by the DefwindowProc() function .

Every Windows application will have a message loop. The loop will contain either the GetMessage() function or a

PeekMessage() function. The GetMessage() gives the control back to Windows until there is any message for it; so that Windows can process other application programs. This is the significance of message loop allowing user to switch between applications. If a Windows program is written without a message loop, unless it terminates immediately, it takes over all of the system's processing time and the user will not be able to switch to another program.

3.2.3 Message processing:

All the messages are normally processed in a function named WndProc(). In order to let the message data from newly registered class to be passed to the WndProc() function, the address of WndProc() function should be visible to Windows. Hence, we have to 'export' the address of WndProc() by adding a line saying "EXPORTS WndProc" to the Definition file of the application. Windows can locate the WndProc() function through what are known as "Task Database" and the "Module Database" tables which will be updated automatically by Windows as soon as the WndProc() function moves in memory. In order to separate the message processing to be done only in WndProc(), in the message loop, we call "Dispatch Message()" function. All the message data incoming will be stored first in MSG data structure and then we get that data through a call to GetMessage() and then hand it over to WndProc() through

DispatchMessage(); where we pass the address of message field of MSG data structure.

Finally, in the WndProc(), to stop an application, we use the WM_DESTROY message received from Windows, and call "PostQuitMessage(0)". Then the message loop receives a WM_QUIT message. The Get Message() on receiving WM_QUIT returns zero which breaks the message loop.

3.3 window Properties :

Another way to associate data with created windows is through window properties. Suppose, there is a window instance for a class that you did not register yourself. You want to associate data with that window. But, since you have not registered the window class your self, you can't appraise the no. of extra bytes specified in the WNDCLASS structure. Even if you can retrieve this information using "GetClassInfo"; the allocated extra bytes may be in use by the window procedure that operates on this class. Using these extra bytes for your own task will definitely interfere with the behaviour of this window.

Properties pave the way for you to associate data with a window by suing a string name instead of modifying the information stored in the internal window structure. Notice that only 16-bit values may be associated with a property.

Windows provides four functions to manipulate properties of a window:

- The SetProp function associates a property with a window.
- The RemoveProp function removes a property associated with a window.
- The GetProp function retrieves the property associated with a window
- The EnumProps function retrieves the list of all properties associated with a window.

3.4 Memory Management :

As the memory is a vital shared resource under Windows, to avoid fragmentation, as soon as new applications crop up and the older ones get finished, Windows does memory management consolidating free memory space by moving blocks of code and data in system memory.

Under Windows, a program's code can be more than the capacity memory at one time. Windows discards code from memory and later reload the code from the program's .EXE file. Routines located in "dynamic link libraries" can be shared by programs running in Windows. Windows links both at run time. Windows itself is a set of dynamic link libraries.

If a big windows program comprises many C or C++ program files, after compiling, each individual C/C++ program results in a separate piece of final program called a "segment". Windows can load and remove each segment of a program to and from memory separately. At any time, there could be any of the segments of all available might be loaded into memory. If a function call is made in presently loaded segment, automatically, the segment possessing the called function will be loaded into memory; before the execution of the calling segment.

Windows manages the data also similarly. Each piece of data in the application program's resource data is handled separately by Windows. As the program runs towards execution, each piece of data referred in the resource data can be set in such a way that it is not loaded into memory until it is needed. So each data item can be separately loaded and unloaded from memory as and when it is needed. For the Windows programmer, there is an Option to choose as to how he wants his program code and data is managed in memory. These options for Memory should be indicated in module definition file and in the resource script file.

Memory Management Options in Windows

OPTION	**MEANING**
PRELOAD	The code or data is loaded into memory when the program first starts.
LOADONCALL	The code or data is not loaded into memory until it is needed.
MOVEABLE	The code or data can be moved in memory. This is the most common option.
FIXED	The code or memory remains at a fixd address in memory. (Usually used in rare situations such as interrupt driven interfaces with hardware devices.)
DISCARDABLE	The code or memory can be temporarily removed from memory to make room for other objects. The code or data will then be reloaded if it is needed.
MULTIPLE	Applies only to program data. MULTIPLE means that if the same

program is started several times (several instances) each will have a different set of data.

Stacks and Heaps:-

Each application program in Windows uses it sown single memory segment for maintaining its "stack" and "local heap". Automatic variables used in the program are stored in the stack, while global and static variables are stored in the local heap. the program's code is in one or more segments. A separate segment holds the stack and local heap. The automatic variables will be over written each time a function body is entered, while global and static variables, occupy fixed positions within the segment.

There can be requests for blocks of memory in Windows programs. These requests can be for Fixed Moveable, or discardable blocks of memory as the program runs.

3.5 Borland Resource Workshop :

(TO CREATE CUSTOM RESOURCES)

The various important Resource Types are:

1. Keyboard Accelerators
2. Bitmaps
3. Cursors

4. Icons

5. Menus

6. Dialog Boxes

We can add more and more custom Resource Data specific to each application consisting of all Windows controls, Dialog Boxes, List Boxes, ComboBoxes etc. very easily and painlessly using Broland's Resource workshop. Once the resource data is available in the resource file, when we make a Turbo C++ protect comprising the main program file, the resource file, the header file if any, and the Module definition file; and compile it the following steps occur. The resource compiler compiles the resource file to make an .RES resource data file. Then the linker takesover. It Combines the data in the .RES file with the program wherever applicable forming a final totally compiled Windows Application.

The Bordand Resource Workship package consists of individual Resource Editors. The BRW is a completely integrated environment designed to run under Windows. The BRW is a Windows program with full menu and dialog Box support. If the "New" option is selected from the File menu of the BRW, several protect types can be selected. By clicking one of the project buttons shown, the user can elect to design icons, cursors, dialog boxes, and more.

3.6 A look at the Module definition file:

The data given in the module definition file specifies the memory options that are specific to Windows.

An example module definition file is as below:

```
NAME            NEW PROGRAM
DESCRIPTION     Project thesis of B.V. Raveendra
EXETYPE         WINDOWS
CODE            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        1024
STACKSIZE       5120
EXPORTS         WndProc.
```

1. The NAME statement specifies the name of the file and is opsional.

2. DESCRIPTION adds a textstring specified to the beginning of the file.

3. EXETYPE always specified as WINDOWS for Versions 3.0 and above. If not specified as WINDOWS, it is taken as WINDOWS 2.0 is the current WINDOWS package version.

4. The STU statement has the name of a small Dos programm WINSTUB.EXE to alert user with a message saying "This program is to be run under Microsoft Windows", if he attempts to run it from DOS.

6. The program's data segment size is set with the HEAPSIZE and STACKSIZE specified.

7. The EXPORTS statment specifies the names of the functions that vill be accessed directly by Windows as soon as the programm runs.

CHAPTER 4

OBJECT WINDOWS CONCEPTS

If we have to reuse portions of programs already written earlier, it will be cumbersome. No doubt, we can always reuse the entire program. But, if we have to take support of small bits and pieces of various programs to make a completely new program, certainly a lot of modification and reprogramming will be required or at times it may be impossible also. The concept of Object-Orientation to create reusable objects comes to our rescue, in this direction. The advantage of Object-oriented programming in this situation is that you can modify parts of an object to change specific things while keeping the rest of the object. Object orientation is very well there inherent in the Windows programming style itself.Windows defines everything on the screen as a window. Each window is in effect an object that inherits some of its features from other windows. The data associated with a window is encapsulated into the window object. polymorphism is another salient feature that results from object orientation. Window objects are polymorphic, they can all receive any common messages and take action that is appropriate for that window.

To make the Windows programming much easier and to reduce the burden to a great extent, we used the Object Windows Library (OWL) that comes with Borland C++ for Windows

(BCW) package. Basically, we have to create an application object and a window object first, as the descendants of TApplication class and TWindow class of the OWL. Over and above that, we go on adding menu objects, dialog objects etc. We are always at freedom to change the parts of objects according to our needs, while navigating the program. A Windows program written without using OWL involves checking for previous instances of the program, registering window classes, and defining window procedures. The usage of OWL simplifies making a Window application by reducing the number of steps needed to create two objects that are descendants of the TApplication and TWindow classes as mentioned earlier.

Let us consider a sample minimal example object windows program and analyze.

```
#include <owl.h>
class TAppname:pubic TApplication
{
public:
      TAppname(LPSTR ApName, HANDLE PevInst,
              LPSTR CmdLine, int CmdShow):
      TApplication (ApName, Inst, PrevInst, CmdLine,
              CmdShow) {};
      virtual void InitMainWindow();
};
```

```
void TAppname::InitMainWindow()
MainWindow=new TWindow(NULL,"MyProgram");
}
int PASCAL WinMain(HANDLE Inst,HANDLE PrevInst,
                   LPSTR CmdLine, int CmdShow)

{
      TAppname Appname("An OOP", Inst,
                          prevInst,CmdLine, CmdShow);

Appname.Run();
return Appname.status;
}
```

The include statement for "owl.h"file is a must for an OWL application, inorder to include the Object Windows objects and constants. The next part of the program defines a descendant of TApplication called TAppname. The definition implies that this class has a constructor that calls the constructor for TApplication and a member function (TAppname) that overloads the InitMainWindow() member function. The next step in the program involves creation of a TWindow object by the "InitMainWindow" function. The string that appears on the caption bar (i.e. "My program", here) is an optional one inputted by the programmer. The final part of the program is the WinMain function that Windows calls when it starts the program. The WinMain function first creates a

TAppname object. Creating this object calls the constructor which setsup the application. Then the WinMain calls the members function 'Run'.This function starts by calling the InitMainWindow member function. It is the InitMainWindow member function, that create and displays the application's main window when we run the program. All the message processing for messages being sent from Windows for moving, repainting, minimizing, maximizing, closing the application window, is done by 'Run'.

Finally, to terminate the application program, when Windows sends a WM_QUIT message to the program, the Run function returns to WinMain which inturn returns the <u>Status</u> variable from the TApplication object to Windows.

In a nutshell, an OWL application's main program normally comprises just three statements.

1. The first statement of the WinMain() constructs the application object, by calling its constructor.

2. The second statement calls the application's 'Run' member function.

3. The third statement returns the final status of the application that object windows stores in 'Status' data member.

In the execution of the program, the sequence of actions that take place are as below:
'Run' calls 'InitApplication' & 'InitInstance' to perform the first instance and each instance initialization, respectively.
'InitMainWindow' is then called to create a main window.
The application then starts moving ahead with the 'Run' member function calling the <u>message loop</u>, to begin processing incoming windows messages.

4.1 A Comparative Study :

In normal Windows Program	In an OWL application program
1. Explicit checking for previous instances of the program.	1. Object-Windows calls InitInstnace for every instance of the application.
2. Explicit call to Register class() function to register the class for an application window under WNDCLASS .	2. No explicit registration for window creation. Object Windows uses a default class for registering the window class. It is the TWindow class normally that does this.
3. Explicit call to CreateWindow() function to create the application window.	3. 'Run' on its own calls Init Main Window() member function that creates a window.
4. Explicit call to the message loop involving GetMessage()function to receive messages for the program. If none are there for the application, GetMessage() returns control back to Windows until it gets a message for application.	4. Inherent message loop structure. Object Windows calls the IdleAction member function of the TApplication class whenever there are no messages for the application.
5. Each time a Windows function is to act, the corresponding window's handle is to be specified	5. The HWindow member function of the window object encapsulates the handle requiring you to specify the handle only once.
6.Normally all the application's message processing is done in a function called WndProc().	6. No need to define a separate Wndproc. Member functions of TApplication class handle messages from Windows.
7.Explicitly the program should process WM_COMMAND message etc. with a switch statement, searching for which command message has come.	7. Indexed referencing for messages eases the task.
8.Default message processing for moving, sizing, closing the application window is handled by calling DefWindowProc().	8. The 'Run' member function does all the default message processing for an application window.
9.For terminating an application,explicit call to DestroyWindow() function is to be made.	9.Just before an application exits, object windows calls the TApplication member function "CanClose" to see if it is OK to stop the application.

This comparision clearly shows how easy it is, to develop a windows program using standard OWL. Borland's OWL provides the following advantages in Windows program development.

A simplified and consistent interface to windows is given. By consistency we mean, knowing how to develop one complete OWL application program makes the programmer being able to create any other Windows program; because all the steps involved in that process use standardized formats supplied by OWL.

For window management and message processing, OWL has several automatic routines built-in which will be called accordingly. For structuring a Windows application, a basic frame work is supplied.

4.2 Features Of Object Windows :

[]. Encapsulation of Window information:

The window object possesses its own member functions which call the various windows functions. Those windows functions act on the windows using individual windows' handles. The window handle is encapsulated within the object's 'HWindow' data member. We need not specify it each time.

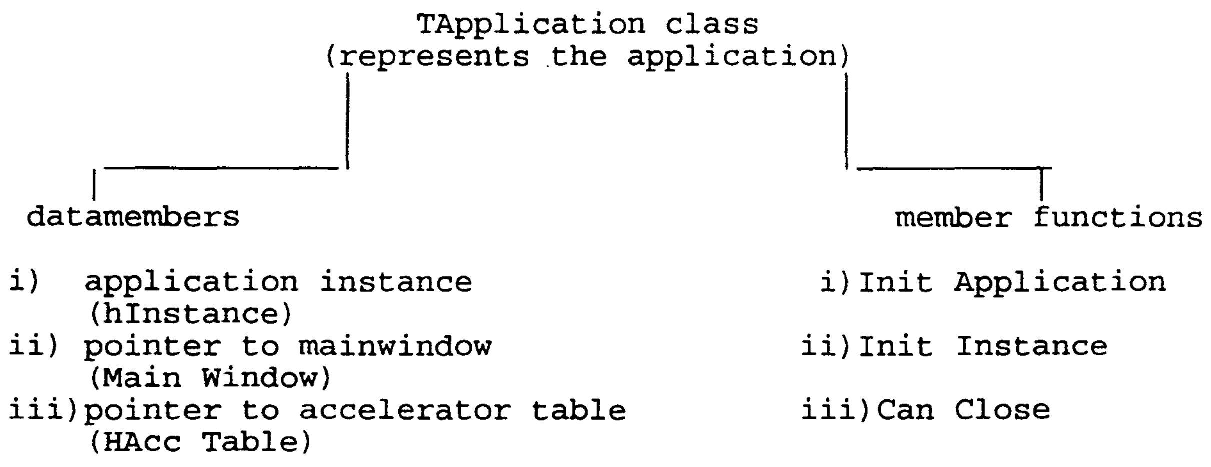
[]. Abstraction of many Windows API functions:

All the related function calls are grouped into single member function by the Object Windows. This results in less dependence on numerous API functions for the same task.

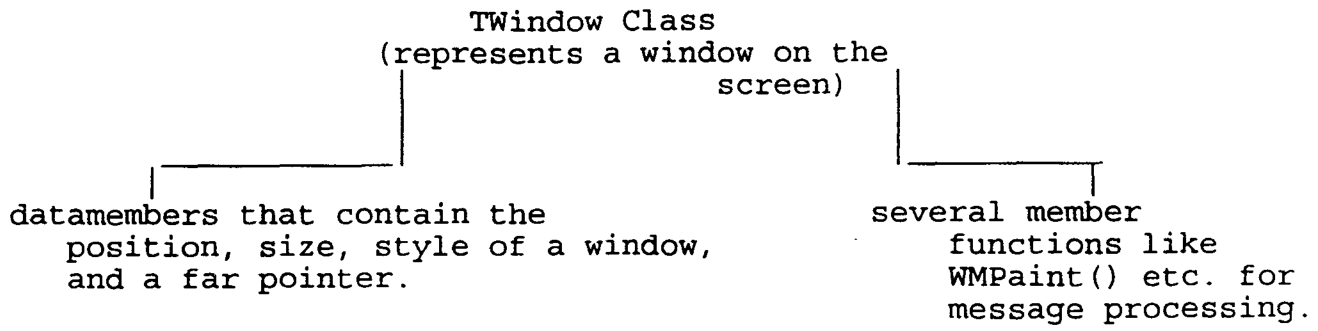
[]. Automatic message response:-

The DefWndProc() function is <u>automatically</u> invoked by the Object Windows for all the messages which are not being processed by the application program. In the case of normal Windows program without using OWL, this needs an explicit call to DefWindowProc() function by the application program.

4.3 Object Windows Class Hierarchy:


TApplication class
(represents the application)
datamembers
member functions
i) application instance (hInstance)
ii) pointer to mainwindow (Main Window)
iii)pointer to accelerator table (HAcc Table)
i)Init Application
ii)Init Instance
iii)Can Close

During a call to 'Run' the 'InitApplication' and 'InitInstance' member functions are called by Object Windows to initialize the application. To stop the application, Object Windows calls the 'CanClose' member function.


TWindow Class
(represents a window on the screen)
datamembers that contain the position, size, style of a window, and a far pointer.
several member functions like WMPaint() etc. for message processing.

A TWindow object is created in the InitMainWindow() function. The constructor for this class sets all the object variables for size, style etc of a window. When the application receives a WM_CREATE message from the Windows for the window creation, then the member functions of this class are activated.

The registration of a window class is done by the TWindow class. Since, we define the application's class as a descendant of TWindow class, due to Inheritance the registration automatically takes place for the application window's class. If the way the window is registered by Object Windows is to be changed, we need two member functions to overload this class. The first one is <u>GetclassName</u>. It returns a pointer to class. The second function is <u>GetWindow class</u>. It fills in the WNDCLASS structure with information about the window class.

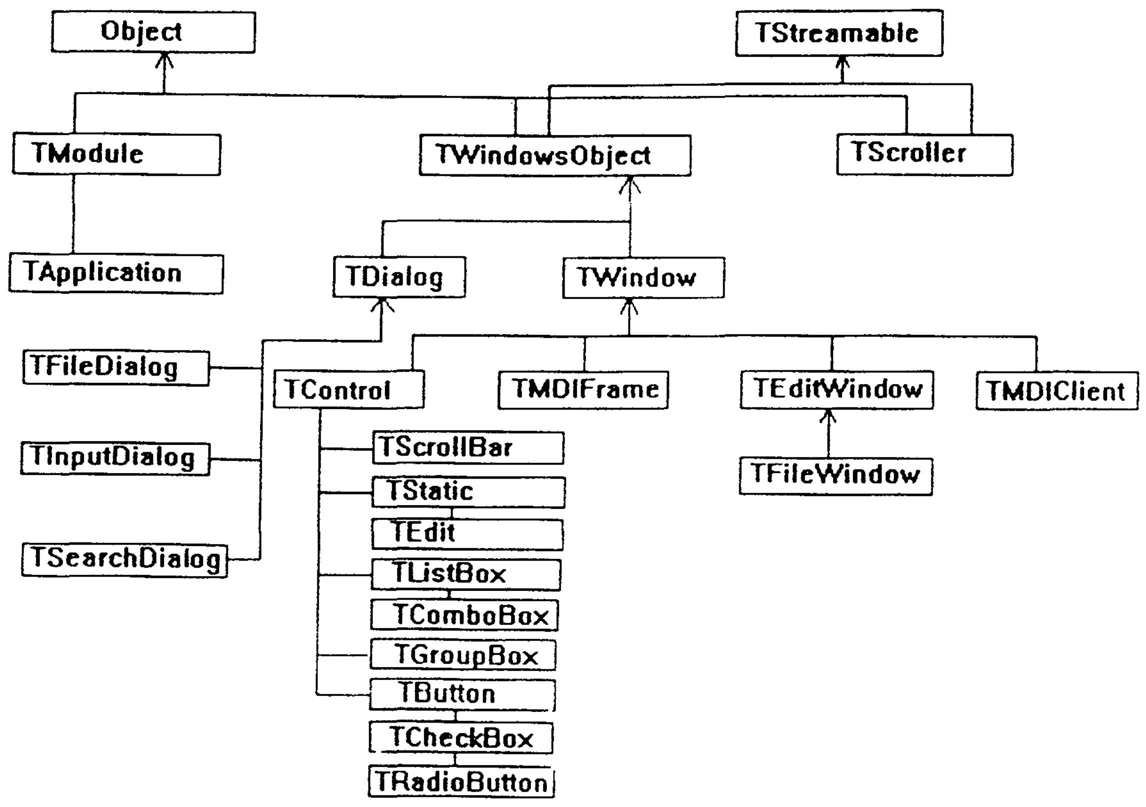
The OWL has definitions for various object classes like TButton, TDialog, TEdit etc. for objects Button, Dialog, Edit etc. respectively. Objects like edit windows are called control windows, and are most frequently used in Dialog boxes. Object Windows treats control windows as descendants of TWindow class. Control windows are always associated with their parent windows. To copy information from more than one control windows, it suffices to give a single function call. For example, the data entered in a dialog box in several edit windows can be copied at a time to storage area by using the TransferData member function. To create Multiple Document Interface (MDI) applications, TMDIClient and TMDIFrame classes are useful.

Borland C++ package provides message response functions to easily define a member function which manages the Windows message. There are several ranges of numbers that object Windows uses for indexed member functions. For example, WM_FIRST leads the index into the range for Windows messages CF_FIRST leads the index into the range for messages from controls etc. The format of a reference message can be as follows:

```
      Virtual void function (RTMessage Msg)
                         =[Index offset + message from Windows];
```

RTMessage implies Reference to TMessage structure.

The TApplication calss is divided from TModule class which itself is derived from Object class.



(FROM OBJECT WINDOWS FOR C++ USER'S GUIDE p 224)

OBJECT WINDOWS HIERARCHY

TModule	: supports Windows memory management and error processing.
TWindowsObject	: supports handle creation, message processing and destruction of windows object.
TStreamable	: defines the behaviour shared by all windows objects.
TWindow	: represents main, pop-up and child windows of an application.
TEditWindow	: allows text editing in a window.
TFileWindow	: In addition to text editing, allows loading and saving text files.
TControl	: defines member functions of handle creation and message processing for all controls.TButton, TList Box, TCheck Box, etc all controls are derived from TControl class to represent the respective controls.
TStatic	: defines member functions that set, query and clear the text of a static control.

TEdit	:	provides text processing capabilities for a window's Edit control.
TFileDialog	:	supports file opening, editing & saving.
TInputDialog	:	defines a dialogbox for user input of a single data item.

CHAPTER 5

DESIGN AND IMPLEMENTATION

5.1 Railway Database Design:

The design involves two parts-the design of the database and the design of the user interface.

A database is an integrated collection of automated data files related to one another in the support of a common purpose. Each file in a database is made of "dataelements". These data elements are to be organized by, stored in, and retrieved from the database. In the view of a database designer, data means automated information, i.e. information in a format acceptable for automatic processing by a computer. For achieving automated information, it should be reduced into a machine-readable form. As we all know, the smallest component of data in a computer is the "bit", a binary element with the only allowed values of zero and one. Bits form the building blocks of bytes (or characters), which inturn are used to build data elements. Data files contain "records" that are made up of data elements, and a database consists of files. So, the heirarchy is as follows:

1. Data base
2. File
3. Record
4. Data element

5. Character (byte)

6. Bit

While designing a database the first five levels in the heirarchy are of importance. That is because, the database contains files of records that contain data elements, made up of characters.

Databases are filled with items of information. The values of these items must be stored somewhere. The places where they are stored are data elements. A data element is a place in a file used to store an item of information that is uniquely identifiable by its purpose and contents. For each application there will be relevant data elements. In this Railway Reservation System, some relevant dataelements used are Origin, Destination, Fare1 (fare in I class), Fare2 (fare in II class), Train name, Train No., Deptime (departure time), Day (Day of the Week on which it is operational), Quota1 (No. of bearths in I Class), Quota2 (No.of bearths in II class), No. of coaches, PNR No. (unique identification no. of a ticket), Distance (in kilometers), Name (of the passenger), Age, Sex, Address, Date (Date of journey) etc. The functional relationship of these data elements to the application's purpose is important. The context of the function supported by the system is contributed to by the data elements recorded in a database. We can also view the data element as a temporary repository for a transient value or the value of the instance of information. The dataelement assumes a specificd data value in a specific instance. Hence, a data value is appropriately defined to be the information stored in a data element.

Notice, that a database is a set of files related to one another by a common purpose. We can say that a file is a set of records where the records have the same data elements in the same format. Each record will have a format which is determined by the order of storage of data elements in a file. After notifying the files, records and data elements we have to define the schema. A "Schema" is the expression of the database in terms of the files it stores, the data elements in each file, the key data elements used for record identification, and the relationships between files. Now, the file keys are to be identified. A file "key" logically points to the record that it indexes. The unique file key that differentiates each record from all others in a file is the "primary key". Only one record in a file can contain a given value of a primary key. Put in otherwords, the primary key data element in a file is the data element used to uniquely describe and locate a desired record. A combination of more than one data element can also form a key. Finally, the inter-relationships among files are to be identified; since we are using a Relational Database in this project.

5.1.1 Database Design steps :

The design of a relational database is carried out as shown below in a sequence of steps.

1. Identify the basis for the database reqiurements.
2. Define the database functional and performance requirements.
3. Identify the data items.
4. Separate the data elements from the files.
5. Build the data element dictionary.
6. Gather data elements to files.
7. Identify the retrieval characteristics of each file.
8. Identify the relaionships between files.
9. Develop the schema for the DBMS you are using.

STEP1: The problem definition forms the basis for the database requriements.

Problem:- Keep track of the vacancies available in various trains to the specified destinations on specified dates. Record the passengers taking part in journey along with their PNR No.'s and the fares calculated for their journey. Report the status of reservations made taking into considertion all the cancellations being made. Report the Trains detials & fares detials. Provide complete help on each operation of the system.

<u>Resources:</u> The list of available resources includes information from

1) The passenger's application form for reservation or concellation.

(2) The standard fares specification table of Railway Department.

(3) The standard trains time-table of Ralway Deparment.

.STEP 2:-

<u>Functional Requirements:</u>

1. The system records the reservation data for each passenger on taking input for Name, Address, Age, Sex, Origin (Station), Destination (Station), Class (of journey), Date (of journey). The data will be stored with the same data element names. The system allots a unique PNR No. for each ticket reserved. The system calculates the correct fare of journey by checking the class and age.

2. The system deletes the data for a particular passenger on taki

the PNR No. for cancellation.

3. The system reports the complete details of ticket booked/reserved, on production of the ticket's PNR No.

4. The system reports the total list of tickets reserved on a particular date.

5. The system records the trains information in records. Each record includes, the train no. train name, the departure time, the origin, the destination, the no. of coaches, the quota 1 (capacity in I class), quota 2 (Capacity in II class), Day (on which operational).

6. The system will reocrds the fares information in records. Each record includes, Origin, Destination, Fare 1 (in I class), Fare 2 (in II class), Distance (in km). The passenger can book a ticket in either I class or II class. In calculating the fare for a passanger, the system checks if his age is below 14 years. If so, the system charges one-half of the fare calculated either in I class or in II class. If the age is above 14 years, the system takes the Fare 1 or Fare 2 as it is, correspondingly.

7. The system reports with a list of trains with each train's record on request from a passenger.

8. The system reports with a list of fares with each fare's records on request from a passenger. The fare to a specific destination can also be calculated.

9. The system gives a provision for the database administrator to take a backup of all the tickets reserved on a particular date. Also everyday morning before starting reservation operations, it builds all the records reserved for a particular date into a doubly linked list and will be ready for the next action of reservation or cancellation etc.

10. The system also provides on optional exit of the menu.

Performance Requiremnets:

1. The system will support upto 45 days prior reservation.

2. The system will support upto 200 stations.

3. The system will support upto 50 trains.

4. Each train consists of 15 coaches.

5. Each first class coach will have 50 beraths capacity.

6. Each second class coach will have 70 breaths capacity.

7. Retrieval of data recorded about a passenger, a train will be on-line and will be in response to the user's entry of the PNR No. of the ticket, the train no. respectively.

STEP3:-

The collection of possible data items reads as follows. (Notice the word "data items" we used is not the data elements list only). Fares, Trains, passanger, Train no. Train Name, Date of journey, Class, No. of Coaches, Distance, Name, Address, Age, Sex, Day (optional), Quota1, Quota2, Fare1, Fare2, PNR No., Origin, Destination, Status, Bearths1, Bearths 2.

STEP4:-

Here, from the above list in step 4, we should separate out the files and the data elements. In terms of the entity-relation model (ER - Model), the entities and their attributes are to be distinguished.

The data elements identified are:

Train No., Train Name, Day (Operating), No. of Coaches, Origin, Destination, Quota 1, Quota 2, DepTime, Class, Date (of journey), Name (of passenger), Addrss, Fare1, Fare2, Age, Sex, PNR No. Bearthes1, Bearths2, Distance.

The relations identified are:

Passenger, Trains, Fares, Status.

Notice: The relations are the files themselves.

STEP5:-

The physical properties of all the dataelements that will be in the database are described in what is known as a "data element dictionary". A data element dictionary is a table data elements including at least the names, data types, and lengths of every data-element in the subject database.

Data element Name	Data Type	Length	Remarks
Name	character string	20	
Address	character string	40	
Age	Integer	4	
Sex	character	1	
Origin	string	10	
Destination	string	10	
Class	Integerter	2	
Date	String	14	
Train No.	Integer	8	
Train Name	String	20	
Day	String	10	
No. of Coaches	Integer	4	
Quota 1	Integer	4	
Quota 2	Integer	4	
Fare 1	Integer	8	
Fare 2	Integer	8	
Distance	Integer	10	
Deptime	Float	10	upto two decimal places
Bearths 1	Integer	4	
Bearths 2	Integer	4	
PNR No.	Integer	8	A four digit number

STEP6:-

As we have identified earlier, the files of the database are passenger, Trains, Fares and Status. Here, we will consider each of them as an entity and try to assign the corresponding attributes to them.See figure2.

There are finally four tables designed.

Passenger Table									
PNR No.	Name	Address	Age	Sex	Train No.	Date	Origin	Desti-nation	Class

Trains Table							
Train No.	Train Name	Deptime	Destination	Distance	No.Of Coaches	Quota1	Quota2

Fares Table				
Train No.	Destination	Class	Fare1	Fare2

STATUS TABLE			
Date	Train No.	Bearths1.	Bearths2

Notice in the Trains table the very first destination is always the Origin of the train, where from it starts.

STEP:7&8:-

In this step, for each file the retrieval characteristics are to be defined. The retrieval of records of information is dependent on what are known as "Primary Keys".

1. For the cancellation of a ticket, the ticket's PNR No. will be the primary key.
2. For the display of a particular ticket's details, its PNR No. will be the primary key.
3. Display the file which lists the complete trains information on request of viewing it.
4. Based on the Train No. Destination and Class, the correpsonding fares details can be known.
5. To list all the Tickets reserved display all the records of passenger relation one by one.

To know the inter-relationships among the files, first let us identify all the relations with their keys of retrieval.

Passenger:	PNR No.
Train :	Train No.
Fares :	Train No. Destination, Class
Status :	Date, Train No.

PNR No., Train No. are the primary keys of relations passenger, and Trains. Train No. together with Destination and Class form the candiate key for the relation Fares.

The functional dependencies assumed for the database design are:

Passenger:	PNR No----> Name	PNR No---->Age
	PNR No----> Address	PNRNo----->Sex
	PRN No---->Train No.	PNR No---->Date
	PNR No---->Origin	PNR No---->Destination
	PNR No---->Class	
Fares:	Train no destination class --->Fare 1	
	Train no. Destination class --->Fare 2	
Status:	Date Train No--->Bearths1	
	Date Train No--->Bearths2.	

STEP 9:-

Defining the database schema invovles, all the practical steps carried out in the implementation of data base.

5.1.2 Railway Database Implementation :

The passenger relation has been stored in TICINFO.DAT File (The file name indicates that ticket information is stored in that file). The train relation has been stored in a file named TRINF.DAT. The Fares relation has been stored in FARE.DAT file. The Status relation has been stored in STATUS.DAT file.

Ticket is a structure defined with all the members to collect information for Name, Age, Sex, Address, etc., with a left pointer to the node and with a right pointer to the node called "prior" and "next" respectively. Also, a pointer to this structure itself called "Info" is defined. A doubly linked list is used to hold the data of the structure members.

After each reservation operation the whole data of that passenger is stored into TICINFO.DAT from the doubly linked list. A structure is defined to input the records into TRINF.DAT containing information for trains. Similarly, FARE.DAT and STATUS.DAT are also filled through structures defined for them.

5.2 User Interface Design

The user interface stands as a mediator between the user and the database.

1. User selects the options and gives the commands. User inputs the appropriate data also.

2. Taking the inputted data, the system searches the database and retrieves the proper records, and operates on that data.

User interface should provide a menu to the user. Since, we are concerned with a Railway Reservation System the manu consists options to reserve, cancel, View ticket details, trains and fares details. To perform the respective tasks, the interface needs relevant data from the user. This data is accepted through a dialog box. If the validity of input data is violated the interface should prompt back to the user with a message box, giving warning.

The user interface should not restrict the user to operate only with the mouse. It should give to the user provision for shortcut key access also. That means, keyboard accelerators are to be incorporated.

The interface design should be such, that at each step, flexibility of decision should be left to the user. That is to say that suppose, in listing all the tickets reserved, there should be two push buttons in a message box with YES and NO options. If the user selects YES, the system should proceed listing next ticket. If the user selects No, the control should go back to the main window menu, clearing the client area. Appropriate help, whereever required should be given.

5.2.1 Main Program Flow :

```
Start
      Initialize TApplication Object datamembers
      Through Inst & PrevInst, perform first and each
      instance initialization
      Process InitMainWindow
      Construct TMainWindow object
      Enter message loop
            {
                process Messages for the applciation, until
                WM_QUIT message does not occur
            }
            End Message loop on finding (WM_QUIT)
            Destory TMainWindow and TApplication objects
            return status
End
```

5.2.2 Construct TMainWindow object

```
Start
      Assign menu to TMainWindow object
      Build the doublylinked list using the passenger
      information file
End
```

5.2.3 Process Main window messages

```
start

      case option selected
            menuitem Reserve           => Execute TReserveDlg
            menuitem Cancel            => Execute TCancelDlg
            menuitem Listtickets       => Process List
            menuitem View  Ticekt      =>  Execute  TViewTicketDlg
            menuitem viewTrains        => Process TrainsDetails
            menuitem ViewFares         => Execute TFaresDlg
            menuitem Save              => Process Save
            menuitem Load              => Process Load
            menuitem Exit              => Process CanClose
            default                    => Default Windows Processin
      End case

End
```

Figure aside shows the message processing for the main window. The corresponding member function is executed by the TMainWindow object, as soon as a message comes. If the user selects the reserve option, the application gets a CM_Reserve message from Windows. The Reserve member function is then executed.

5.2.4 Execute TReserveDlg

```
Start

     Construct TReservDlg's child objects
     Show Reserve dialog box
     Fetch the data into various buffers to hold
     name, origin, destination etc.

Case  option selected

       OK button : If (!( Valid Train no & Valid  origin &
                    Valid destination))
                    Give error message
                   else
                    {
                   if (datebuffer >(date of booking + 45))
                   Give  a  message  saying  out  of
                   reservation period

                  else
                   {
                    Search the STATUS file and check for
                    vacancy.
                  if (vacancy exists)
                       {
                        PNRno.=PNRno+1
                        Allot PNRno.
                        Get fare1  or fare2 from FARES file
                        depending on class is first or second
                       if (age <14)
                       fare = fare/2
                       Create  a new node in the doubly
                        linked list and load the whole data
                       if (class is first)
                           Quota1=(Quota1)-1
                       else
                           Quota2=(Quota2)-1
                       Store new Quota in STATUS file
                       Display  the details of ticket booked
                        along with the PNRno. allotted
                       Close TReservaDlg and
                       return to TMainWindow
                       }
                       else
                       Give a Message saying no vacancy
                     }
                  }

      Cancel button         :  Default Windows processing
             default        :  Default Windows processing

End case
End.
```

5.2.5 Execute TCancelDlg

```
Start

      Construct TCancelDlg's child objects
      Show cancel dialog box
      Fetch the PNRno. inputted into a buffer called
      numbuffer
      Case option selected
           OK button: If(!(Valid PNR no)) give error message
                      else
                      {
                       Traverse all the nodes in the linked
                       list for a match with the PNR no. in
                       numbuffer

                       If a match occurs, delete the
                       corresponding node from
                       the linked list
                       Release the PNRno.
                       If (Class is first)
                          Quota1=(Quota1)+1
                        else
                           Quota2=(Quota2)+1
                           Store new Quota in STATUS file
                           Give a message to the user that his
                           ticket is cancelled.
```

```
                    fare=(fare-10/100*fare)

                    Output the fare calculated

                    Close TCancelDlg and return to
                    TMainWindow

                }

        Cancel button : Default Windows processing

              default: Default Windows processing

  End case

End
```

5.2.6 Process List

```
Start

   Case option selected

        OK button:

              Initialize the pointer to the first node of
              ticket data
              repeat
                 Display the ticket data held by node
                 Message box with YES & No option asking if
                 the user wants to see next ticket or not

              until (IDNO)
              return to TMainWindow
```

```
                Cancel button : Default Windows processing
                       default: Default Windows processing
     End case
End.
```

5.2.7 Execute TViewTicketDlg

```
Start
Show view ticket dialogbox
Fetch the PNR No inputted into a buffer called numbuffer
Case option selected
              OK button: if (!Valid PNR No.)
                            Give error message
                            else
                               {
                               Move the pointer of the list to
                               point to first node
                                while ((node-->PNRno.)!=(numbuffer))
                                {
                                Goto next node;
/*last is apointer which always    points to the    end of
last node in list*/
                                if(list pointer ==last)
                                     set flag;
                                 }
```

```
                              If (flag)
                              {
                              Give a message that ticket not found
                              Close TViewTicketDlg
                              return to TMainWindow
                              }
                           else
                              {
                              Display the whole node data
                              Close TViewTicketDlg
                              return to TMainWindow
                              }
                            }

                  Cancel button : Default Windows processing
                       default: Default Windows processing
          End case
     End
```

5.2.8 Process Trains Details

```
Start

 Give a messageBox with YES & No option asking if the user
wants to see the whole list of trains.

  case option selected
      OK button: Create a child window
```

```
         Open the TRAINS file
         Display it in the child window
         return to TMainWindow

      Cancel button : Default Windows processing
             default: Default Window processing

  End case

End
```

5.2.9 Execute Fare details

```
Start
      Construct TFaresDlg's child objects
      Show Fares dialog
      Fetch origin, Destination and class

      case option selected
           OK button : If!(Valid destination)
                            Give error message
                         else
                            {
                            Search FARES file and retrieve
                            the record with the given
                            destination and class
                          if (class is first)
                             Display (fare1)
```

```
                              else
                                Display (fare2)
                                Close TFaresDlg
                                return to TMainWindow
                                }

                 Cancel button : Default Windows processing
                        default: Default Windows processing
      End case
End
```

5.2.10 Process Save

```
Start
      Open a file in appendmode
      save the whole linked list's data in that file
      prompt back to user with a message box

End
```

5.2.11 Process Load

```
     Start
        open the file used for saving
        Build the doubly linked list with all the nodes data
        from the file
        prompt back to user with a message box

     End
```

5.2.12 Process Exit

```
    Start

            Store the linked list contents in TICINFO file
            Save the STATUS file
            DestroyWindow()
End
```

CHAPTER 6

SAMPLE SESSION

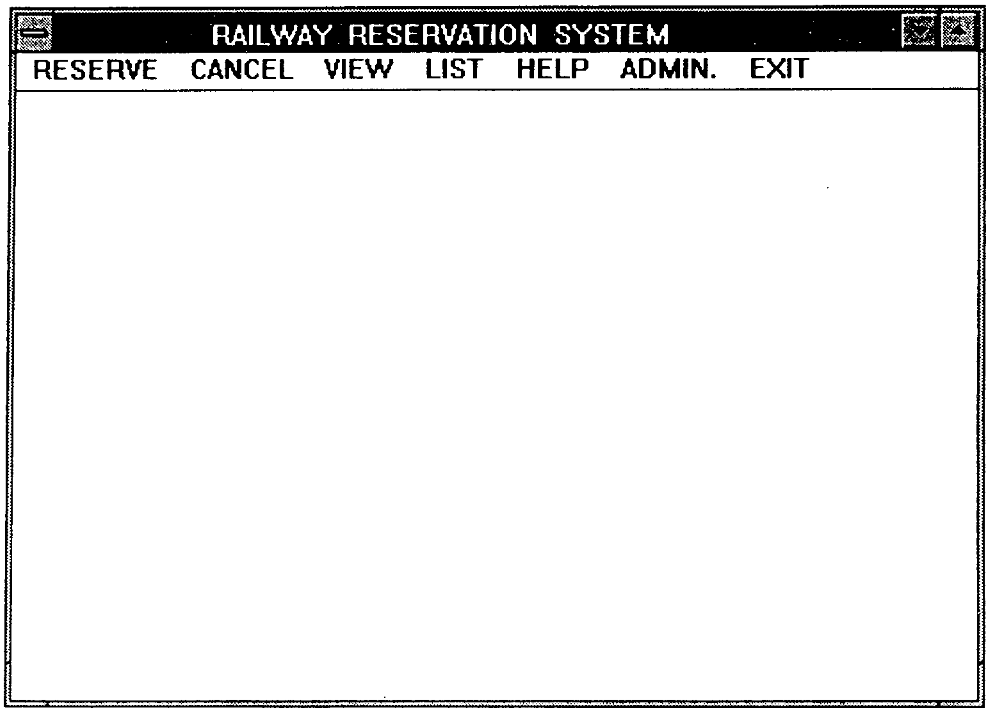
A project file is created using the main program file, the resource file, the header file for the program and the module definition file. Using Borland C for Windows (BCW), compile the project file. From the program manager run the application program. Then the application's main window appears on the screen. This is shown in the figure in page 68 .

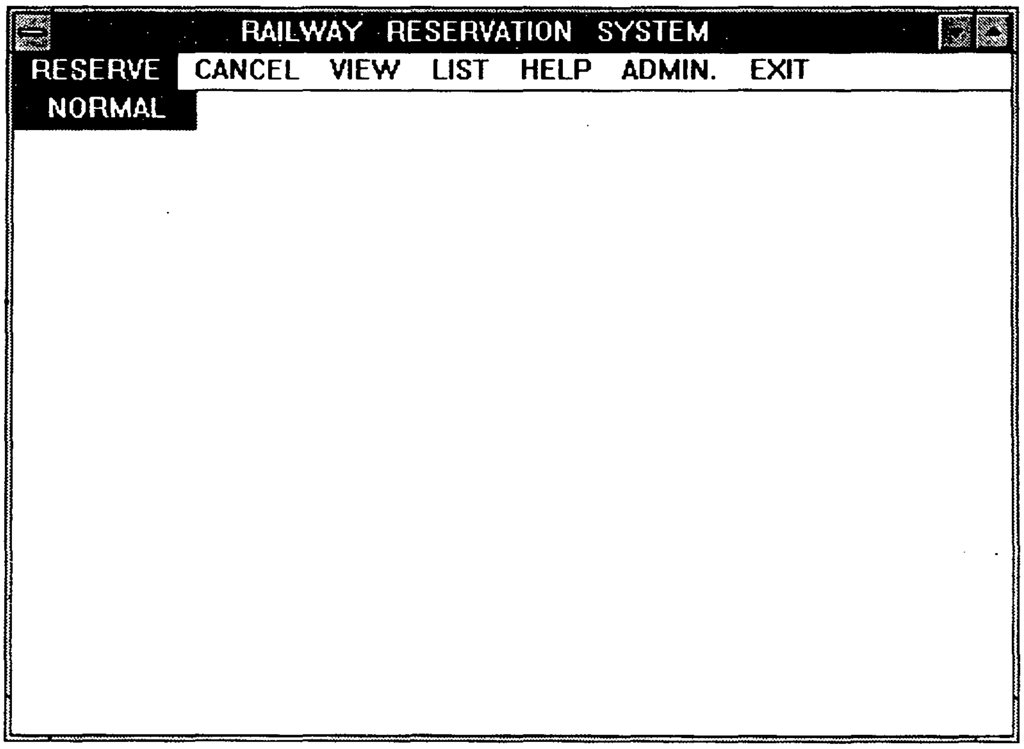
This contents Reserve, Cancel, View, List, Admin, Help, Exit options. The various popup menus are as follows :

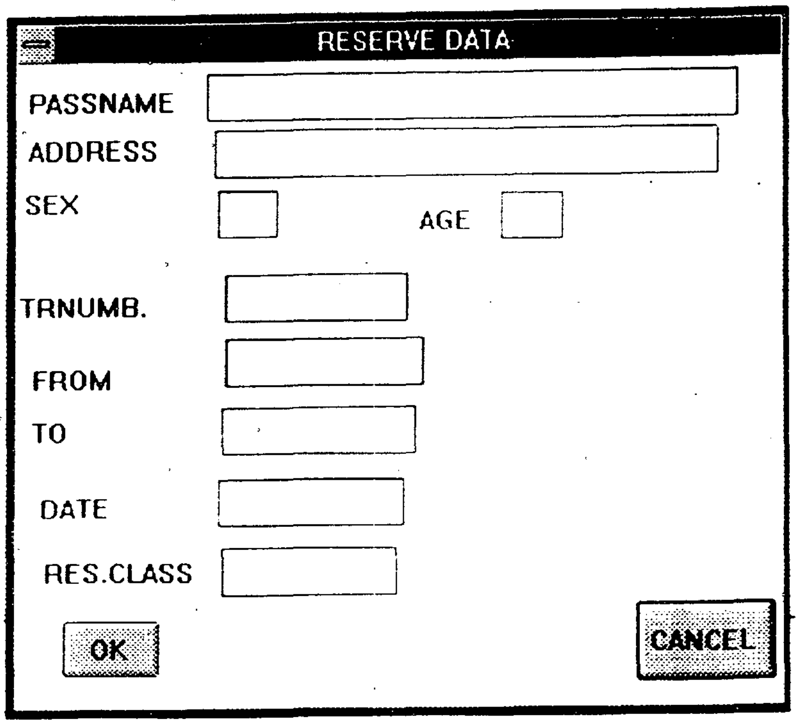
Menu Item	Popup Menu	Remarks
Reserve	Normal	Gives, TReserveDlg Dialog box.
Cancel	Normal	Gives TCancelDlg Dialog box.
View	- Ticket	Gives TViewTicketDlg Dialog box.
	- Trains	Displays Trains Information file.
	- Fares	Displays Fares Information file and TFaresDlg.
List		List all the tickets booked.
Admin	- Save	Saves into a file
	- Load	Loads into a linked list from file.
Help		Provides Help.
Exit	Quit menu	Quits the application.

The TReserveDlg is shown in the figure in page 70 . The TCancelDlg is shown in the figure in page 72 The TViewTicketDlg is shown in the figure in page 74 The List gives output as shown in figure in page 76. The Save Popup menu on selection gives a message box as shown in figure in page 78 , while saving the ticket's information into TICINFO.DAT file. The Load popup menu on selection gives the message shown in figure in page 79, after loading the ticket's details into a linked list. The Quit menu popup menu for Exit menu option is shown in the figure in page 80.

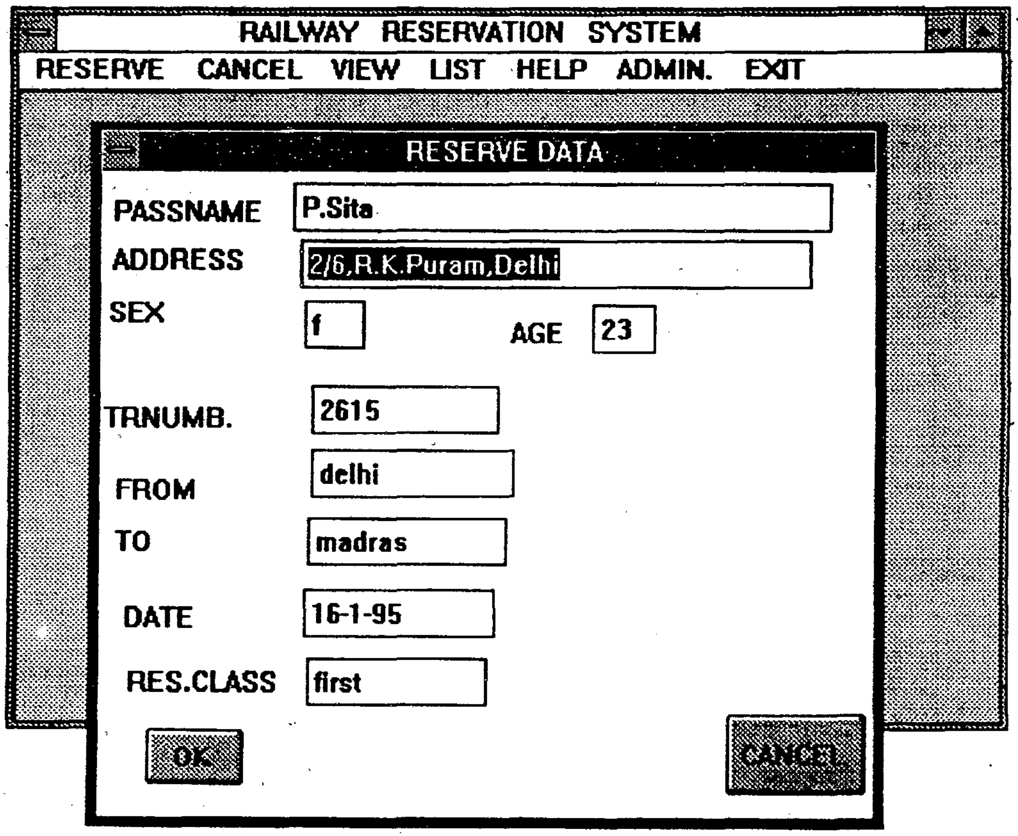
For all data inputs, data validation checking is carried out.


RAILWAY RESERVATION SYSTEM
RESERVE
CANCEL
VIEW
LIST
HELP
ADMIN.
EXIT


RAILWAY RESERVATION SYSTEM
RESERVE
CANCEL
VIEW
LIST
HELP
ADMIN.
EXIT
NORMAL

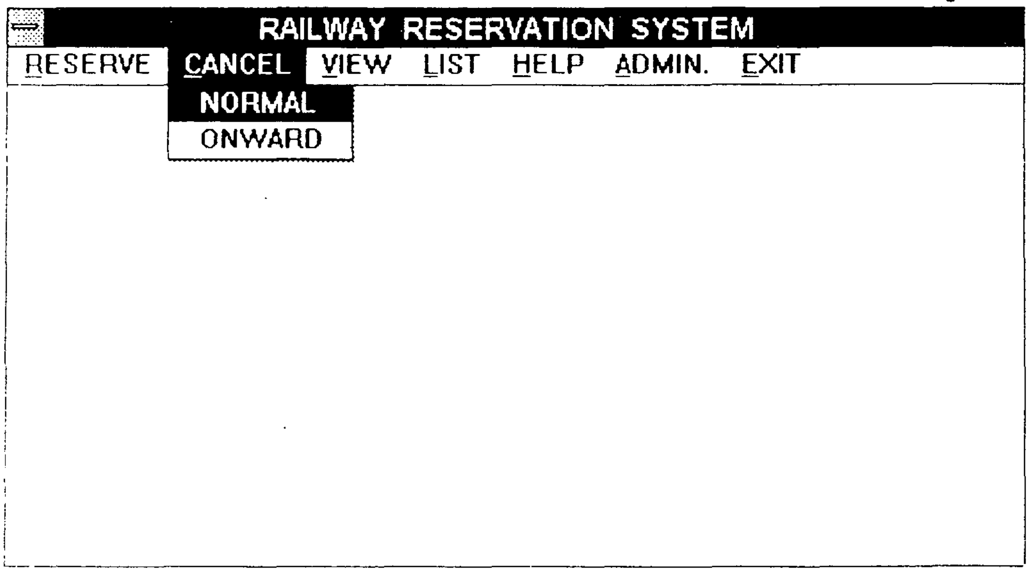


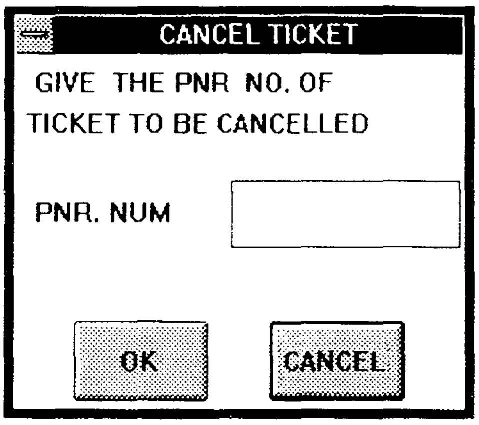
PASSNAME	
ADDRESS	
SEX	
AGE	
TRNUMB.	
FROM	
TO	
DATE	
RES.CLASS	



PASSNAME	P.Sita
ADDRESS	2/6,R.K.Puram,Delhi
SEX	f
AGE	23
TRNUMB.	2615
FROM	delhi
TO	madras
DATE	16-1-95
RES.CLASS	first

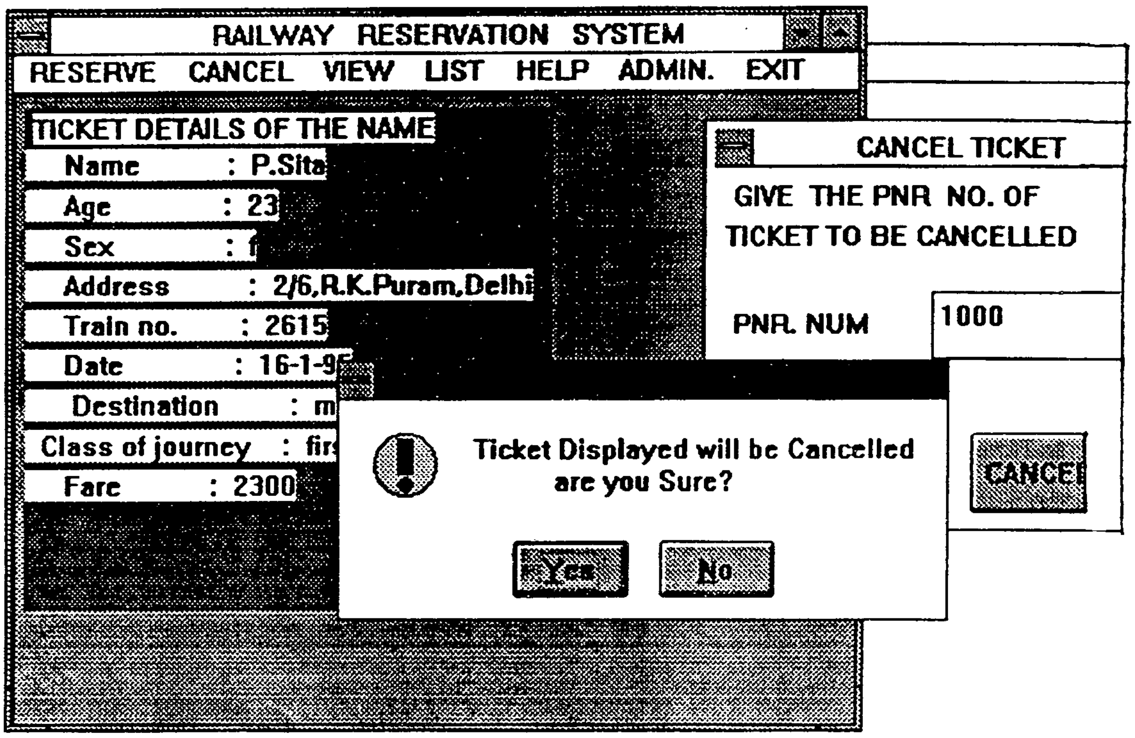
RAILWAY RESERVATION SYSTEM
RESERVE CANCEL VIEW LIST HELP ADMIN. EXIT
RESERVE DATA
OK
CANCEL


RAILWAY RESERVATION SYSTEM
RESERVE
CANCEL
VIEW
LIST
HELP
ADMIN.
EXIT
NORMAL
ONWARD



GIVE THE PNR NO. OF
TICKET TO BE CANCELLED

PNR. NUM



RAILWAY RESERVATION SYSTEM

RESERVE CANCEL VIEW LIST HELP ADMIN. EXIT

TICKET DETAILS OF THE NAME

Name : P.Sita

Age : 23

Sex : f

Address : 2/6,R.K.Puram,Delhi

Train no. : 2615

Date : 16-1-9

Destination : m

Class of journey : fir

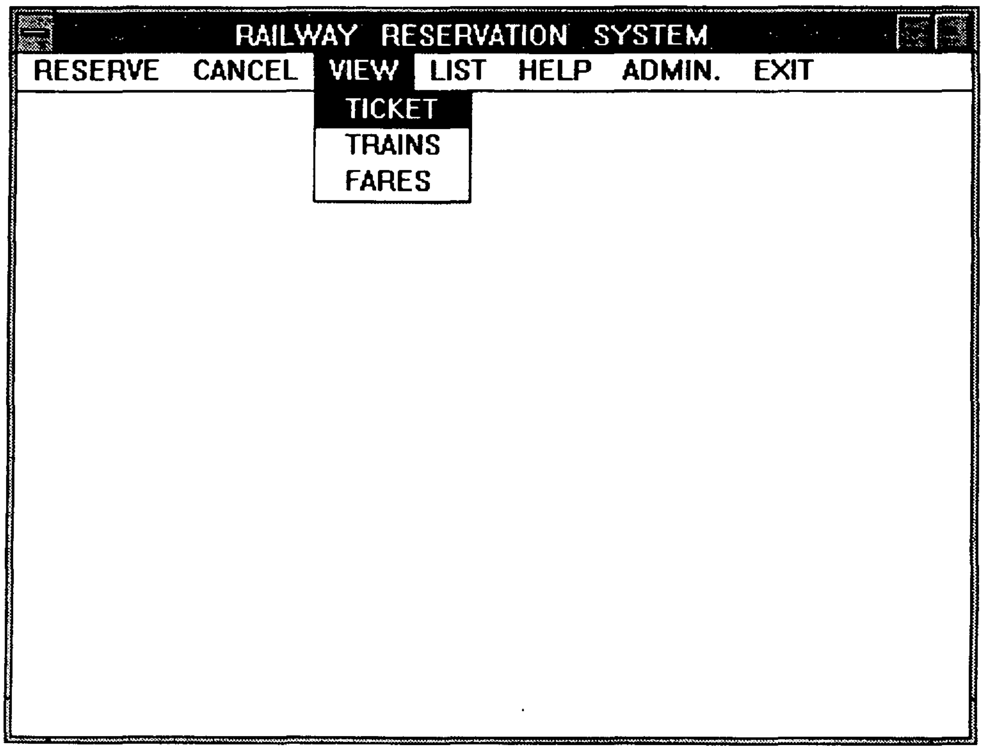
Fare : 2300

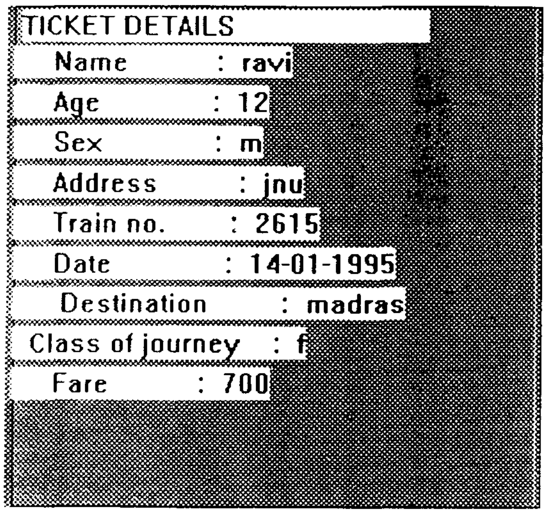
CANCEL TICKET

GIVE THE PNR NO. OF TICKET TO BE CANCELLED

PNR. NUM 1000

Ticket Displayed will be Cancelled are you Sure?


RAILWAY RESERVATION SYSTEM
RESERVE
CANCEL
VIEW
LIST
HELP
ADMIN.
EXIT
TICKET
TRAINS
FARES


TICKET DETAILS

Name : ravi

Age : 12

Sex : m

Address : jnu

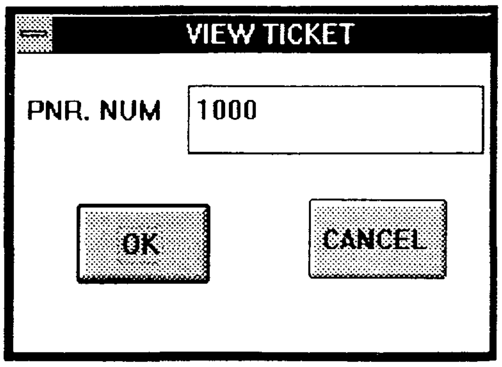
Train no. : 2615

Date : 14-01-1995

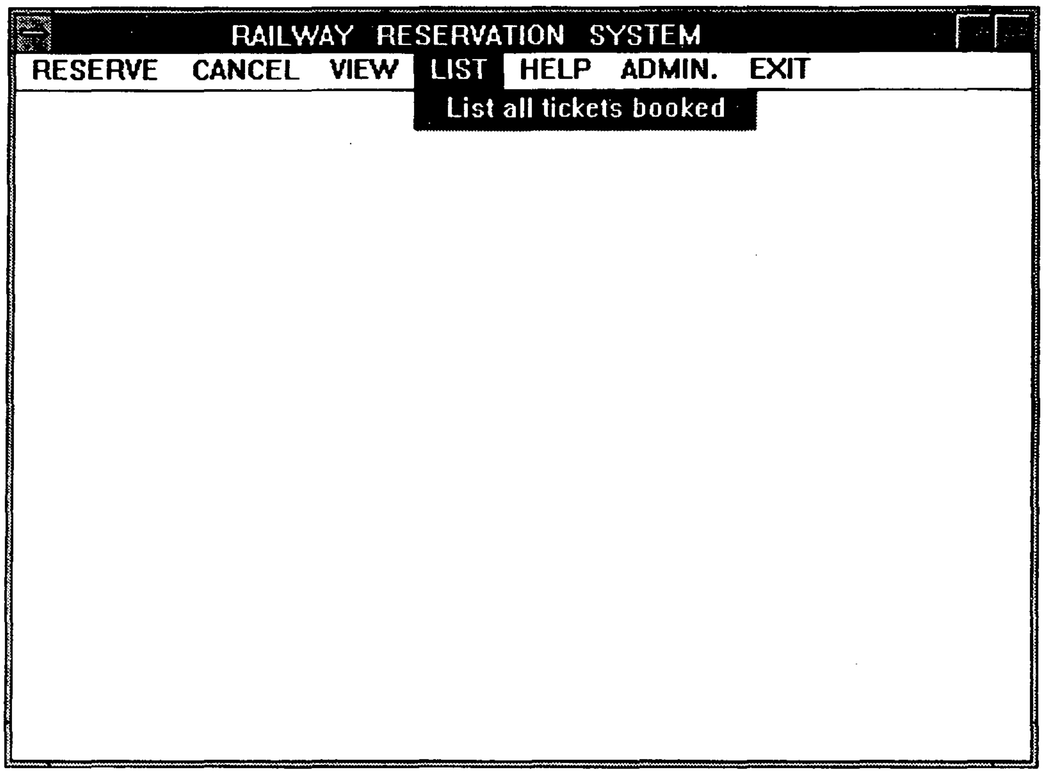
Destination : madras

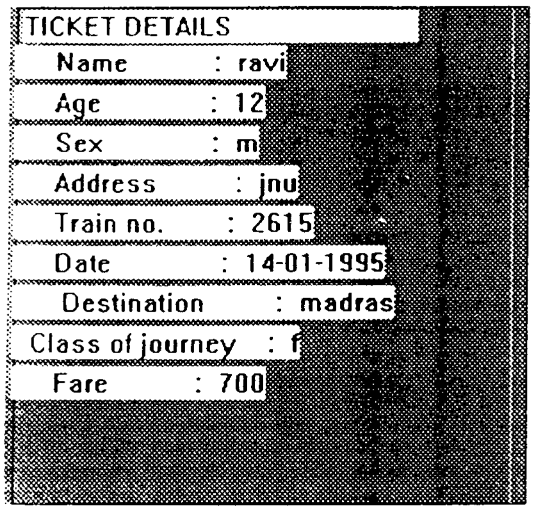
Class of journey : f

Fare : 700


VIEW TICKET

PNR. NUM 1000


RAILWAY RESERVATION SYSTEM
RESERVE
CANCEL
VIEW
LIST
HELP
ADMIN.
EXIT
List all tickets booked



TICKET DETAILS

Name : ravi

Age : 12

Sex : m

Address : jnu

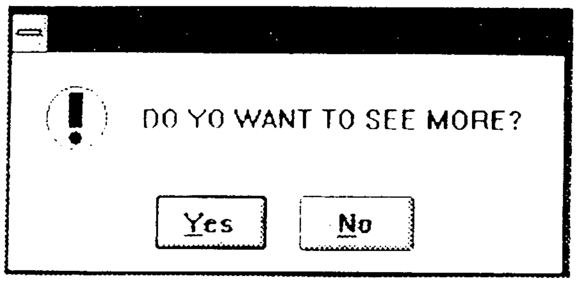
Train no. : 2615

Date : 14-01-1995

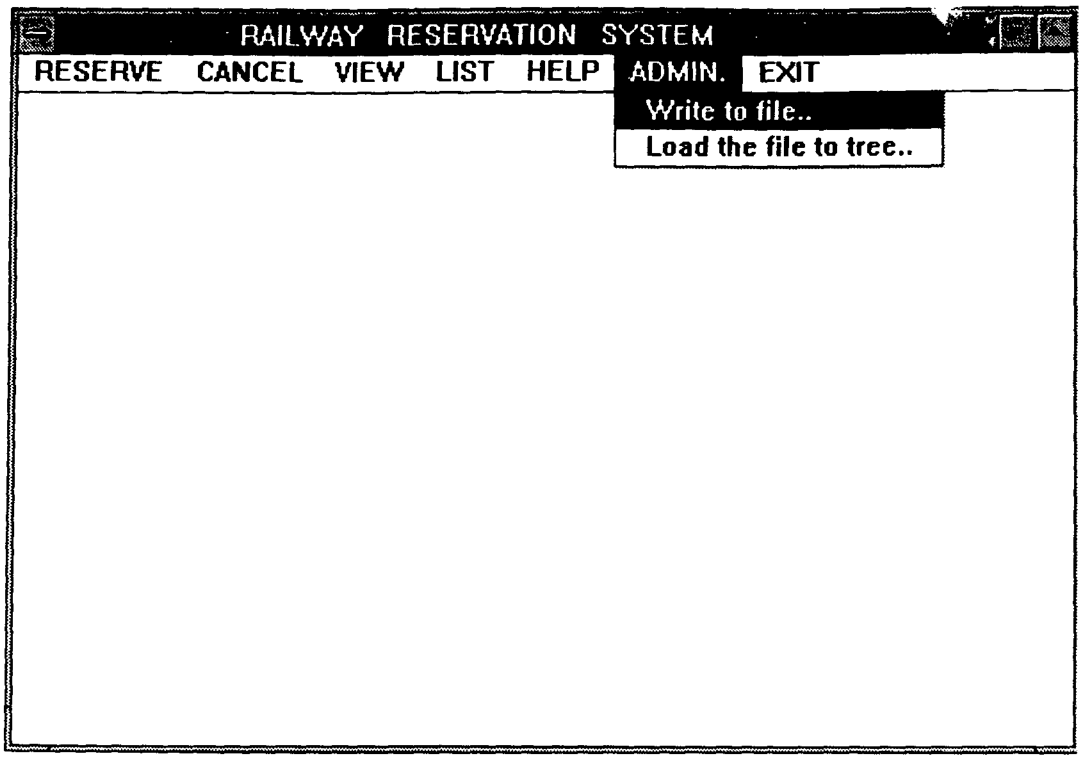
Destination : madras

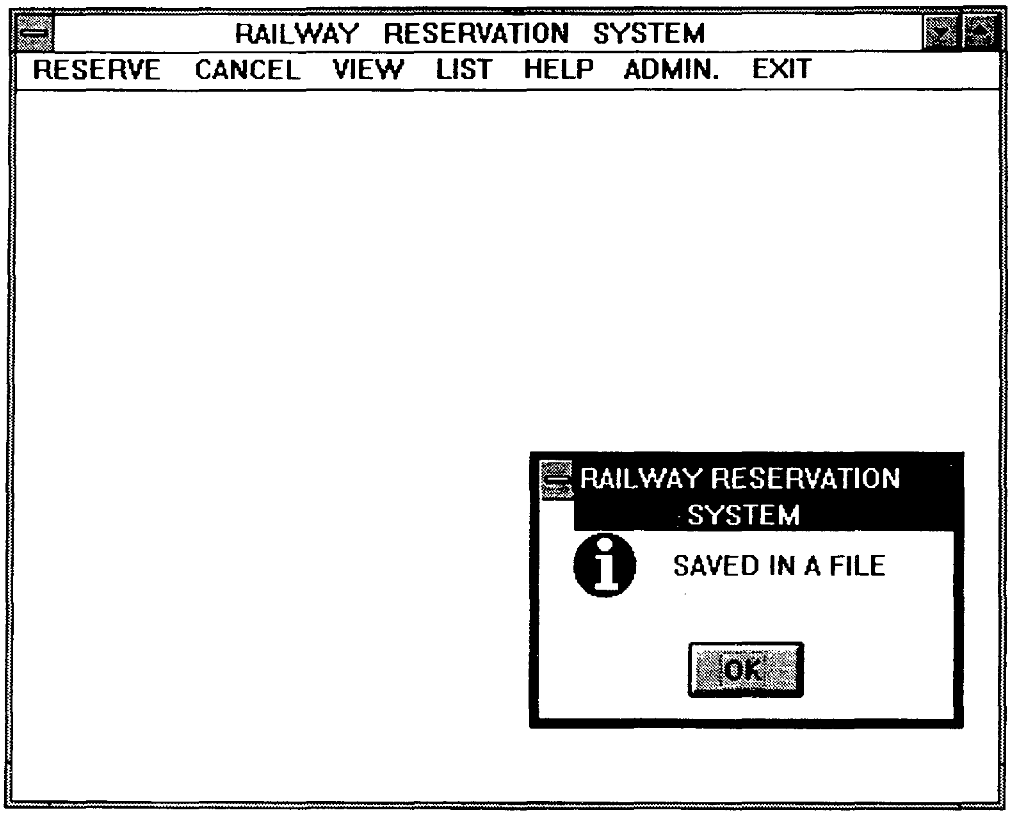
Class of journey : f

Fare : 700

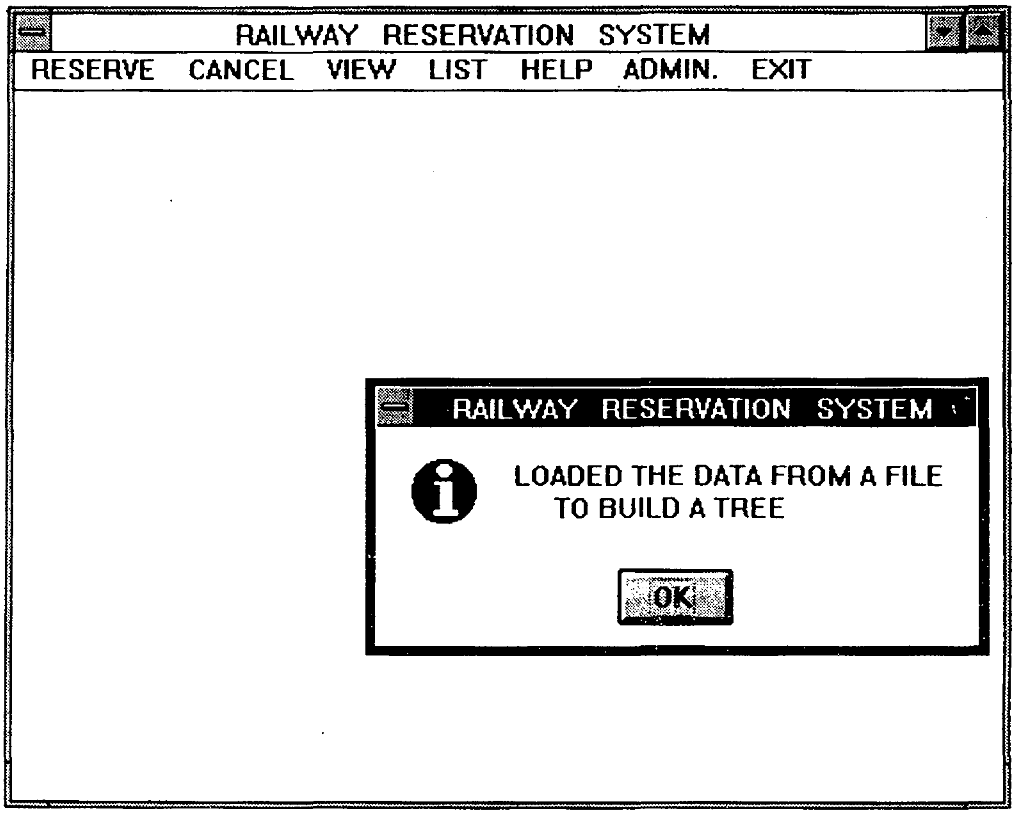


DO YO WANT TO SEE MORE?


RAILWAY RESERVATION SYSTEM
RESERVE
CANCEL
VIEW
LIST
HELP
ADMIN.
EXIT
Write to file..
Load the file to tree..



SAVED IN A FILE

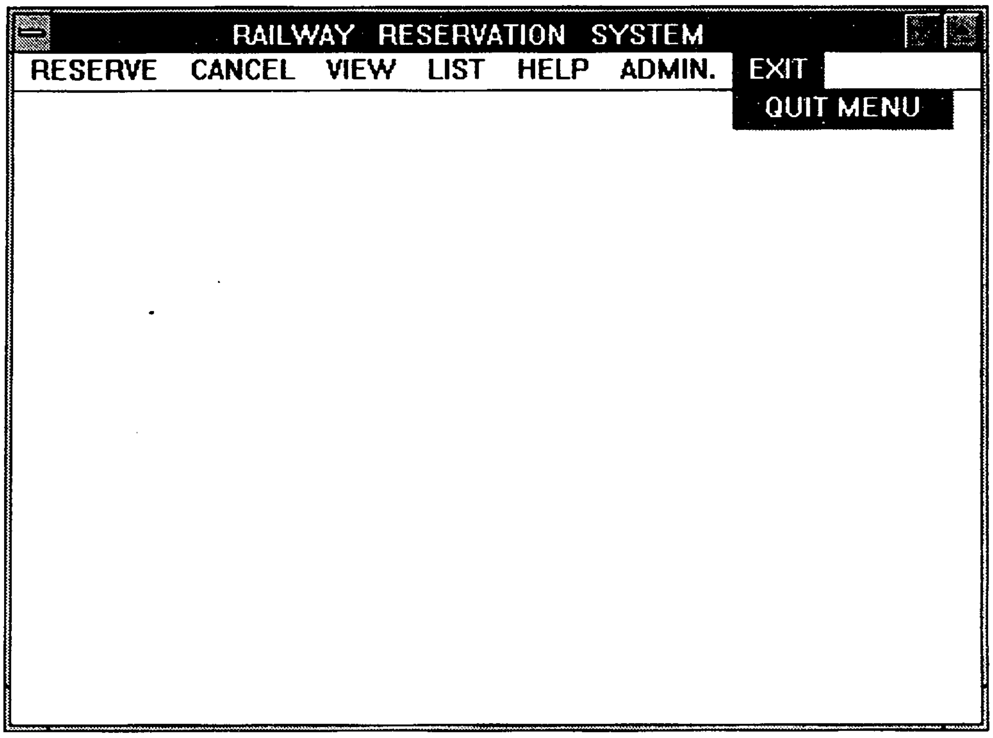

RAILWAY RESERVATION SYSTEM

RESERVE CANCEL VIEW LIST HELP ADMIN. EXIT

RAILWAY RESERVATION SYSTEM

LOADED THE DATA FROM A FILE
TO BUILD A TREE

OK


RAILWAY RESERVATION SYSTEM
RESERVE
CANCEL
VIEW
LIST
HELP
ADMIN.
EXIT
QUIT MENU

CHAPTER 7

CONCLUSION

The design and implementation of a prototype of an object ogiriented GUI for a Railway Reservation System has been carried out in this project, using Borland C++ for Windows and OWL. The work has progressed in two stages. First, the database and then the GUI are designed. Then finally they are implemented. The application also assists the user by giving complete help on how to use it. The fulfledged realworld Railway reservation system is a huge one and complex too. The constraints of limtied time prevented us from implementing a real world database.

Some of the improvements that can be made to this application are:

One of the major improvements that can be made is making the system completely distributed. Over a communication network if the stations are connected, reservation can be made from any station to any other station. Also onward reservation facility can be added to this application.

The provision for preponement and postponement of a reserved ticket can be given. For preponement, marking the PNRno.s of all the tickets cancelled, the same no.s can be again allotted for a preponement request with just moving the "prior" and "next" pointer of the doubly linked list. Similarly, the postponement involves moving the prior and next pointers of a node ahead in the linked list, by incrementing the PNRno. by one, for each movement ahead of a node.

The databse organization can be made more powerful and efficient by using B-Trees for holding the data for searching the database.

As an improvement that can be done in future, a more powerful interface can be accomplished using Windows 4.0, when it enters market. As reported by SteveFox in "computers and communications", October '94, advantages of it are as below:

1. Improved Interface: The Program manager and Filemenager are replaced by "My Computer", "Explorer" and "Start": They make data a point-and-click affair.

2. CONFIG.SYS, and AUTOEXEC.BAT are so well hidden that the programmes or the user need not worry about them at all.

3. Long life names: The limitation on filenames to be "eight dot three" in length is no more there.

4. Clever right mouse button: Right-Click any where to bringup a context sensitive menu.

5. Preemptive multitasking: A new order governs your applications, and a 32-bit architecture provides performance and stability.

6. Plug and Play: If IRQS got you down in version 3.1, Windows 4.0 gives solution. Windows instantly recognizes plug and play cards and peripherals. Plug in and get on with your computing.

7. If an application hangs, you can shut it down without rebooting Windows.

8. Shortcuts: Easy-to-create icons provide quick access to files or applications.

9. Mobile computing: Drag and drop files into your Brief case folder for quick portability and file syrchronization when you return.

10. Communications: Remote network access, and TCP/IP support for interactive inter-net connections, are all standard.

...

BIBILOGGRAPHY

1. Jim Conger, "Windows Programming Primer Plus", Galgotia publications private limited, 1994.

2. Louis Fernades, Yogesh Seth and Anish Kurup, "Borland c++ 3.0 for Windows 3.1 Programming with Object Windows", BPB publications, 1993.

3. Henry F. Korth and Abraham Silbeischatz, "Database System Concepts", Second edition McGraw-Hill Computer Science Series, 1991.

4. Gary Syck, "Object Windows How-To", Galgotia publication private limited, 1993.

5. Brain W. Kernighan and Dennis M. Ritchie, "The C programming Languague", Printice Hall of India private limited, 1993.

6. "Object Windows for C++ User's Guide", Borland International Inc

7. Robert Lafore, "Object Oriented Programming in Turbo C++", Galgotia Publication private limited, 1993.

8. James Conger "Windows API Bible", Galgotia Publications Private Limited,1993.

9. Bjarne Stroup Stroup, "The c++ Programming Language", Addision Wesley Publishing Company, 1994.

10. C.J. Date, "An Introduction to Database Systems" Vol. 1 Adsdison-Wesley narosa Indian Student Edition, 1987.

11. Kaare Christian,"Borland C++ Techniques and Utilities", ZBPress, 1994.

12. Chapman Hall "Object Oriented Programming Systems, Tools and Applications" London 1991.