NUMERICAL SIMULATIONS AND CASE STUDIES USING VISUAL C++.NET

SHAHARUDDIN SALLEH
Universiti Teknologi Malaysia

ALBERT Y. ZOMAYA
University of Sydney

STEPHAN OLARIU
Old Dominion University

BAHROM SANUGI
Universiti Teknologi Malaysia
NUMERICAL SIMULATIONS
AND CASE STUDIES USING
VISUAL C++.NET
NUMERICAL SIMULATIONS AND CASE STUDIES USING VISUAL C++.NET

SHAHARUDDIN SALLEH
Universiti Teknologi Malaysia

ALBERT Y. ZOMAYA
University of Sydney

STEPHAN OLARIU
Old Dominion University

BAHROM SANUGI
Universiti Teknologi Malaysia
CONTENTS

Preface xi

1 Developing Applications Using Visual C++.Net 1
   1.1 Object-Oriented Approach to Visual C++.Net 1
   Managed Extension Features 2
   1.2 MFC Fundamental Features 3
   Graphics Device Interface 4
   GDI Functions 5
   Numerical Functions 6
   1.3 Writing Applications Using MFC 6
   Creating a New Project 7
   Creating a Window 8
   1.4 Writing the First Nonwizard Program 9
   1.5 Discussion 16
   Windows Creation Process 16
   1.6 Summary and Conclusion 17
   Bibliography 17

2 Interfaces for Numerical Problems 19
   2.1 Visualizing a Numerical Problem 19
   The Art of Visualization 20
   2.2 Handling Arrays 22
   Dynamic Memory Allocation 23
   2.3 Finding the Root of a Nonlinear Equation 24
Code2A: Bisection Iterative Method 26
   Code2B: Manual Approach to the SLE Problem 33
Code2C: Resource File Approach for SLE 38
   2.4 Solving a System of Linear Equations 31
   2.5 Summary and Conclusion 48
   Bibliography 49

Bibliography 17
3 Matrix Operations Using Wizard

3.1 Document/View Architecture Using Wizard
3.2 Matrix Algebra
   Data Passing Between Functions
   Matrix Multiplication
   Finding the Inverse of a Matrix
   **Code3A**: Matrix Operations
3.3 System of Linear Equations Problem Revisited
   **Code3B**: Solving the SLE Problem Using Wizard
   **Code3B**: Discussion
3.4 Bibliography

3.3 System of Linear Equations Problem Revisited

Code Listings

**Code2A**: Bisection Method
**Code2B**: Solving a System of Linear Equations
**Code2C**: Resource File Approach to the SLE Problem

3 Matrix Operations Using Wizard

3.1 Document/View Architecture Using Wizard
3.2 Matrix Algebra
   Data Passing Between Functions
   Matrix Multiplication
   Finding the Inverse of a Matrix
   **Code3A**: Matrix Operations
3.3 System of Linear Equations Problem Revisited
   **Code3B**: Solving the SLE Problem Using Wizard
   **Code3B**: Discussion
3.4 Bibliography

3.4 Summary and Conclusion

Bibliography

Code Listings
<table>
<thead>
<tr>
<th></th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code5A:</strong> Mathematical Curves</td>
<td>130</td>
</tr>
<tr>
<td>Drawing a Polynomial</td>
<td>134</td>
</tr>
<tr>
<td>Drawing a Lemniscate</td>
<td>135</td>
</tr>
<tr>
<td>Drawing Creative Net</td>
<td>137</td>
</tr>
<tr>
<td>5.4 Cubic Spline Interpolation</td>
<td>139</td>
</tr>
<tr>
<td><strong>Code5B:</strong> Constructing a Cubic Spline</td>
<td>141</td>
</tr>
<tr>
<td>5.5 Summary and Conclusion</td>
<td>147</td>
</tr>
<tr>
<td>Bibliography</td>
<td>148</td>
</tr>
<tr>
<td>Code Listings</td>
<td>148</td>
</tr>
<tr>
<td><strong>Code5A:</strong> Mathematical Curves</td>
<td>148</td>
</tr>
<tr>
<td><strong>Code5B:</strong> Natural Cubic Spline</td>
<td>152</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>6 Working with Images</strong></td>
<td>157</td>
</tr>
<tr>
<td>6.1 Handling Images</td>
<td>157</td>
</tr>
<tr>
<td>6.2 Bitmap File Format</td>
<td>158</td>
</tr>
<tr>
<td>Raster Operations Involving Bit Shifting</td>
<td>159</td>
</tr>
<tr>
<td><strong>Code6A:</strong> Demonstrating Bit Shifting</td>
<td>161</td>
</tr>
<tr>
<td>6.3 Edge-Detection Problem</td>
<td>166</td>
</tr>
<tr>
<td>Sobel Filtering Method</td>
<td>168</td>
</tr>
<tr>
<td>Laplacian Filtering Method</td>
<td>169</td>
</tr>
<tr>
<td><strong>Code6B:</strong> Detecting the Edges of an Image</td>
<td>170</td>
</tr>
<tr>
<td>6.4 Summary and Conclusion</td>
<td>173</td>
</tr>
<tr>
<td>Bibliographical Note</td>
<td>173</td>
</tr>
<tr>
<td>Code Listings</td>
<td>173</td>
</tr>
<tr>
<td><strong>Code6A:</strong> Working with Colors</td>
<td>173</td>
</tr>
<tr>
<td><strong>Code6B:</strong> Edge Detection Problem</td>
<td>175</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>7 Visualizing a Graph</strong></td>
<td>179</td>
</tr>
<tr>
<td>7.1 Elementary Graph Concepts</td>
<td>179</td>
</tr>
<tr>
<td>7.2 Graph Visualization Model</td>
<td>183</td>
</tr>
<tr>
<td><strong>Code7A:</strong> Drawing a Graph</td>
<td>183</td>
</tr>
<tr>
<td>7.3 Minimum Spanning Tree Problem</td>
<td>191</td>
</tr>
<tr>
<td>Kruskal’s Algorithm</td>
<td>193</td>
</tr>
<tr>
<td>Prim’s Algorithm</td>
<td>194</td>
</tr>
<tr>
<td><strong>Code7B:</strong> Visualizing the Minimum Spanning Tree</td>
<td>195</td>
</tr>
<tr>
<td>7.4 Summary and Conclusion</td>
<td>203</td>
</tr>
<tr>
<td>Bibliography</td>
<td>204</td>
</tr>
<tr>
<td>Code Listings</td>
<td>204</td>
</tr>
<tr>
<td><strong>Code7A:</strong> Drawing a Graph</td>
<td>204</td>
</tr>
<tr>
<td><strong>Code7B:</strong> Minimum Spanning Tree</td>
<td>208</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>8 Graph Applications</strong></td>
<td>213</td>
</tr>
<tr>
<td>8.1 Graph–Network Relationship</td>
<td>213</td>
</tr>
<tr>
<td>8.2 Shortest-Path Problem</td>
<td>214</td>
</tr>
</tbody>
</table>
## CONTENTS

Dijkstra’s Algorithm 215
The Floyd–Warshall Algorithm 216
**Code 8A:** Shortest-Path Visualization 217

8.3 Mesh Network Application 227
**Code 8B:** Shortest Path on a Mesh Network 229

8.4 Summary and Conclusion 234
Bibliography 235
Code Listings 235
**Code 8A:** Shortest Path in a Graph 235
**Code 8B:** Shortest Paths in a Mesh Network 240

### 9 Multiprocessor Scheduling Problem 247

9.1 Parallel Computing Systems 247
9.2 Task Scheduling Problem 249
Task Scheduling Concepts 252
Path Maximum Magnitude Scheduling Model 253
9.3 Task Scheduling Visualization Model 254
9.4 Summary and Conclusion 270
Bibliography 271
Code Listings 271
**Code 9:** Task Scheduling Using Four Processors 271

### 10 Discrete-Event Simulation 281

10.1 Concepts of Simulation 281
10.2 Simulation Model Development 282
10.3 Discrete-Event System Simulations 283
10.4 Multicounter System with Blocking 286
Our Scheduling Model 286
**Code 10A:** Simulating Multicounter Systems 288
10.5 Queueing Systems 295
M/M/1 Queueing System 295
M/M/C Queueing System 297
**Code 10B:** Simulating the M/M/C System 299
10.6 Summary and Conclusion 308
Bibliography 309
Code Listings 309
**Code 10A:** Discrete-Event Simulation with Blocking 309
**Code 10B:** M/M/C Queuing without Blocking 313

### 11 Modeling Wireless Networks 321

11.1 Wireless Cellular Networks 321
11.2 Channel-Assignment Problem 323
11.3 Channel Assignments: Discrete Model 325
CONTENTS

**Code11A:** Channels for Mobile Users 326
11.4 Solving the Channel-Assignment Problem 336
**Code11B:** Solving the Channel-Assignment Problem 336
11.5 Summary and Conclusion 342
Bibliography 343
Code Listings 343
**Code11A:** Channel Assignments to Mobile Users 343
**Code11B:** Channel-Assignment Problem 349

Index 355
Numerical computing has come a long way since the discovery of the first computer in the early 1940s. Computers have brought changes to the world through their capability to handle and solve problems that were previously not solvable. Because of the improvements in information and communications technology that computers have brought, the world looks smaller. There have been vast improvements in the way computers are used for solving numerical problems in which hardware and software together form the building blocks.

This book has been written to discuss both problems in numerical methods and simulations, and their solutions using Visual C++. There are several objectives for doing this. First, there is a gap between a problem and its computing elements. A problem normally comes from a practitioner, whereas the computing elements are the work of a programmer. A practitioner understands the problem and its manual solution well but may encounter problems in implementing the idea through programming. A programmer, on the other hand, has strong analytical skills for programming but may be lacking in providing the expected solution to the problem as it is not his or her area of expertise. As a result, students end up buying some books on Visual C++ and another few books on numerical methods, but still have problems in combining both. A bridge is needed to close this gap. Most books on the market discuss either Visual C++ or the problem exclusively, not both of them at the same time. There are many books specializing in numerical methods and simulations but almost none of them connect to the resources in Visual C++, particularly the Microsoft Foundation Class, or MFC, library. Only a handful of them discuss the problems using the standard C++. For example, the problem of solving a system of linear equations is a fundamental problem in numerical methods. Unfortunately, there are no known books on the market that discuss this problem in depth, especially using the rich resources in MFC. MFC has a large collection of library functions for serving many requirements in programming, but the absence of books in the numerical simulations area may reduce its audience. Today, C++ is facing stiff competition from other languages such as Visual Basic and Java. There is also a relatively new language, C#, which has been mentioned to take over from C++. The issue here is no longer an option, it is survival. Therefore, to remain competitive,
MFC needs to be promoted so that it takes care of areas such as numerical simula-
tions. We have looked at this deficiency and present this book as a solution.

Our second objective is to discuss a problem and its solution and present the so-
lation in a friendly manner. Visualization is the keyword here. A problem will re-
main a problem as long as its friendliness form is not there. It takes time for people
to appreciate a given problem if there is no tool to present its solution in a friendly
manner. Today’s requirements are very challenging to a problem solver. A numeri-
cal solution that displays only a series of numbers will not be appreciated anymore,
not like in the time when FORTRAN dominated the programming world in the
1960s and 1970s. Instead, the solution must be presented in the form of charts,
graphs, animation, and, sometimes, multimedia. Not only that, the interface on the
computer should be friendly to the user when the use of windows, dialog boxes,
mouse, menus, and images are necessary.

Our third objective is to put more emphasis on the problem and try to minimize
the coding using MFC. This is necessary since an approach that involves too many
codes often distracts the reader from understanding the method for solving the
problem. We embark on this idea by implementing the nonwizard approach in MFC
for most problems. The wizard, or guided approach, is presented in one chapter to
compliment the nonwizard approach. Only one application class is involved in the
program design in most examples. This is necessary in order to reduce the complex-
ity in coding. The advantage of our approach can be seen from the small number of
codes required in each application. This benefits the reader, as small lines of codes
make the solution easier to understand. The interface for each problem has also
been designed to be as simple as possible for this purpose. We do not add things
like animation and sound to a typical numerical problem as this approach may be
overreacting. Instead, some relevant things such as edit boxes and a list view win-
dow will be more practical for this application. The problem itself may require a
lengthy discussion and coding. Also, the idea behind this approach is to have the
readers understand the solution to each problem and use the method in their work.
The minimum coding provided in each example will serve as a good beginning for
the reader. It is expected that the readers will pick up the code and expand it in their
real work.

The book is not intended for use as a beginning text book for learning MFC. The
concepts of MFC are not discussed in depth in this book because this is not our
main objective. There are dozens of books on the market today that provide lengthy
discussions on MFC, and we do not wish to compete against them. However, the
MFC concepts related to the topics discussed are explained. Also, it is assumed that
the reader has acquired some programming skills using C++ and understands the
object-oriented approach to programming prior to using this book. This is necessary
since MFC requires some understanding of concepts such as inheritance, polymor-
phism, and overloading.

The programming work in this book has been developed wholly using Microsoft
Visual C++.Net version 2002. The code is also compatible with version 2003 of the
software and Microsoft Visual C++ version 6. The topics discussed in this book
consist of several selected numerical methods and simulation problems. We chose
problems that are fundamental in nature, and ones that will benefit a large audience. The topics range from trivial problems such as the fourth-order Runge–Kutta problem to something quite tricky such as the multiprocessor scheduling problem. We anticipate that the audience for this book will mostly be third-year undergraduate and beginning graduate students. The topics discussed are intended to help students develop their projects at the final-year undergraduate level, intermediate Masters, and beginning Ph.D. degree courses. The book is also suitable for use by practitioners, working professionals, researchers, and lecturers working in the simulation areas.

The work in this book is the result of some years of collaborative research and teaching between Universiti Teknologi Malaysia, University of Sydney, and Old Dominion University. Many materials in this book were developed by the first author for the SSM 3323 and MSM 5023 classes at the Department of Mathematics, Universiti Teknologi Malaysia. The authors would like to thank Professor Ariffin Samsuri, Dean of the Research Management Center at Universiti Teknologi Malaysia, for his support in completing this book. Special thanks also to Michael Till and his group at the CISCO Internetworking Unit, School of Information Technologies at the University of Sydney, Australia; and Kurt Maly, head of the Computer Science Department at Old Dominion University in the United States.

**Shaharuddin Salleh**

**Albert Y. Zomaya**

**Stephan Olariu**

**Bahrom Sanugi**

April 2005
CHAPTER 1

DEVELOPING APPLICATIONS USING VISUAL C++ .NET

1.1 OBJECT-ORIENTED APPROACH TO VISUAL C++ .NET

An object is an instance of a class. A class is a set of entities that share the same parent. Object-oriented programming is a programming approach based on objects. C++ is one of the most popular object-oriented programming languages in the world. The main reason for its popularity is due to the fact that it is a high-level language but, at the same time, it runs as powerfully as the assembly language. In addition, C++ has its roots in ANSI C, which has been a very nicely crafted procedural language, popular in the 1970s and 1980s. But the real strength of C++ lies in its takeover from C to move to the era of object-oriented programming in the late 1980s. This conquest provides C++ with the powerful features of the procedural C and an added flavor for object-oriented programming.

The original product from Microsoft consists of the C compiler that runs under the Microsoft DOS (disk operating system), and it has been designed to compete against Turbo C, which was produced by the Borland Corp. In 1988, C++ was added to C and the compiler was renamed Microsoft C++. In early 1989, Microsoft launched the Microsoft Windows operating system, which includes the Windows API (Application Programming Interface). This interface is based on 16 bits and supports the procedural mode of programming using C. Improvements were made over the following years that include the Windows Software Development Kit (SDK). This development took advantage of the API for the graphical user interface (GUI) applications with the release of the Microsoft C compiler. As this language is procedural, the demands in the applications required an upgrade to the object-oriented language design approach, and this contributed to the release of the Microsoft
C++ compiler. With the appearance of the 32-bit Windows API (or Win32 API) in early 1990s, C++ was reshaped to tackle the extensive demands on Windows programming and this brought about the release of the Microsoft Foundation Classes (MFC) library. The library is based on C++ and it has been tailored with the object-oriented methodology for supporting the application architecture and implementation.

The Net platform refers to a huge collection of library functions and objects for creating full-featured applications both on the desktop and the enterprise Web. The classes and objects provide support for friendly user interface functions like multiple windows, menus, dialog boxes, message boxes, buttons, scroll bars, and labels. Besides, the platform also includes several tedious task-handling jobs like file management, error handling, and multiple threading. This platform also supports advanced frameworks and environments such as Passport, Windows XP, and Tablet PC. The strength of the Net platform is obvious in providing the Internet and web enterprise solutions. Web services include information sharing, e-commerce, HTTP, XML, and SOAP. XML, or Extensible Markup Language, is a platform-independent approach for creating markup languages needed in a web application.

**Managed Extension Features**

A new approach in Visual C++.Net is the Managed Extension, which performs automatic garbage collection for optimizing the code. Garbage collection involves the removal of memory and resources not used any more in the application, which is often neglected by the programmer. The managed extension is a more structured way of programming, and it is now the default in Visual C++.Net. Central to the .Net platform is the Visual Studio integrated development environment (IDE). It is in this platform that applications are built from a choice of several powerful programming languages that include Visual Basic, Visual C++, Visual C#, and Visual J++.

In addition, IDE also provides the integration of these languages in tackling a particular problem under the .Net banner. Visual C++.Net is one of the high-performance compilers that make up the .NET platform. This highly popular language has its roots in C, improved to include the object-oriented elements, and now, with the .Net extension, it is capable of creating solutions for Web enterprise requirements. A relatively new language called Visual C# in the .Net family was developed by combining the best features of Visual Basic visual tools with the programming power of Visual C++.

In addition to its single-machine prowess, Visual C++.Net presents a powerful approach to building applications that interact with databases through ADO.NET. This product evolved from the earlier ActiveX Data Objects (ADO) technology, and it encompasses XML and other tools for accessing and manipulating databases for several large-scale applications. This feature makes Visual C++.Net an ideal tool for several Web-based database applications.
1.2 MFC FUNDAMENTAL FEATURES

MFC is a library that consists of more than 200 classes. Each class has more than a dozen member functions that handle tasks ranging from a simple text display to the more challenging web data manipulation. The MFC library is arranged in a hierarchical manner, as shown in part in Figure 1.1. This hierarchy makes possible a class to derive common member functions from its predecessor classes, thus eliminating redundancies in the classes. The hierarchy also identifies the ranking of each class with respect to other classes, where a class in a higher level is a base class to the given class. In addition, the hierarchy system makes possible further extensions to several new functions of a class, and for the addition of new classes or removal of some obsolete classes. In other words, MFC has been designed in a very modular form so that its future releases will cater to the programming needs of the time.

One of the highest-ranking classes in MFC is CObject. This class is responsible for several general duties, particularly for supporting handle runtime, serializa-

![Figure 1.1 Hierarchy of some selected classes in MFC.](image-url)
tion, and performing diagnostic output for several derived objects. It is from this base class that several other classes are derived, as shown in Figure 1.1.

**Graphics Device Interface**

The GDI, or *Graphics Device Interface*, is a layer in the Windows architecture that insulates the application from direct interaction with the hardware. A class that is commonly derived from `CObject` is the `CDC` class. In MFC, the `CDC` class is the base class for providing an interface with other classes, including `CPaintDC`, `CClientDC`, and `CWindowDC`. Each of these classes makes use of GDI to provide all the basic graphical and drawing functions for an application on Windows through an object abstraction called device context.

A *device context* is a data structure that is responsible for displaying text and graphics as output on Windows. The tools in the data structure are represented as graphic objects such as pens, brushes, fonts, and bitmaps. In reality, a device context is a logical device that acts as an interface between a physical device (such as the monitor and printer) and the application. A device context is a set of tools or attributes for putting text and drawing graphics on the screen using GDI functions.

There are four types of device contexts in GDI: display context, memory context, information context, and printer context. A *display context* supports operations for displaying text and graphics on a video display. Before displaying text and graphics, a display context links with MFC functions for creating a pen, brush, font, color palette, and other devices. A *memory context* supports graphics operations on a bitmap and interfaces with the display context by making it compatible before displaying the image on the window. An *information context* supports the retrieval of device data. A *printer context* provides an interface for supporting printer operations on a printer or plotter.

In Windows, everything including text is drawn as a graphics object. This is made possible as every text character and symbol is formed from pixels that may vary in shapes and sizes. This facility allows flexibility in the shape of the text by allowing it to be displayed from a selection of dozens of different typefaces, styles, and sizes. Text and graphics are managed by GDI functions that are called on every time a graphic needs to be displayed on the screen.

A device context object is created from one of the classes as listed in Table 1.1. For example, the device context `(dc)` in the main window is obtained by deriving this object from `CPaintDC`, as follows:

```cpp
CPaintDC dc(this);
```

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPaintDC</td>
<td>Device context for the server area in Windows</td>
</tr>
<tr>
<td>CClientDC</td>
<td>Device context for the client area in Windows</td>
</tr>
<tr>
<td>CWindowDC</td>
<td>Device context for the whole window</td>
</tr>
<tr>
<td>CMetaFileDC</td>
<td>Device context for representing a Windows metafile, or a device-independent file for reproducing an image</td>
</tr>
</tbody>
</table>
This object can then be linked with the available functions for displaying text, drawing lines, circles, rectangles, and so on. Some of the primitive objects for displaying text and graphics in Windows are pen, brush, font, bitmap, and color palette; these are described briefly in Table 1.2.

### GDI Functions
There are dozens of GDI functions for displaying text and graphics. Table 1.3 describes some of the most commonly used GDI functions for displaying text and graphics. These functions are derived from the CDC class. Basically, a pen is a GDI device for drawing a line. The object is created from the class CPen. The default pen consists of a solid black line with a thickness of one pixel. This shape can be modified by changing the parameters in the class’s constructor, CPen(). A brush is another GDI device for painting and filling a region using the current color. By

#### Table 1.2 GDI objects for text and graphics

<table>
<thead>
<tr>
<th>GDI object</th>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pen</td>
<td>CPen</td>
<td>To draw a line, rectangle, circle, polyline, etc.</td>
</tr>
<tr>
<td>Brush</td>
<td>CBrush</td>
<td>To brush a region with a color</td>
</tr>
<tr>
<td>Color palette</td>
<td>CPalette</td>
<td>Color palettes for pens and brushes</td>
</tr>
<tr>
<td>Font</td>
<td>CFont</td>
<td>To create a font for the text</td>
</tr>
<tr>
<td>Bitmap</td>
<td>CBitmap</td>
<td>To store a bitmap object</td>
</tr>
</tbody>
</table>

#### Table 1.3 Commonly used functions in the CDC class

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc()</td>
<td>Draws an arc</td>
</tr>
<tr>
<td>BitBlt()</td>
<td>Copies a bitmap to the current device context</td>
</tr>
<tr>
<td>Ellipse()</td>
<td>Draws an ellipse (and a circle)</td>
</tr>
<tr>
<td>FillRect()</td>
<td>Fills a rectangular region with the indicated color</td>
</tr>
<tr>
<td>FillSolidRect()</td>
<td>Creates a rectangle using the specified fill color</td>
</tr>
<tr>
<td>GetPixel()</td>
<td>Gets the pixel value at the current position</td>
</tr>
<tr>
<td>LineTo()</td>
<td>Draws a line to the given coordinates</td>
</tr>
<tr>
<td>MoveTo()</td>
<td>Sets the current pen position to the indicated coordinates</td>
</tr>
<tr>
<td>Polyline()</td>
<td>Draws a series of lines passing through the given points</td>
</tr>
<tr>
<td>Rectangle()</td>
<td>Draws a rectangle according to the given coordinates</td>
</tr>
<tr>
<td>RGB()</td>
<td>Creates color from the combination of red, green, and blue palettes</td>
</tr>
<tr>
<td>SelectObject()</td>
<td>Selects the indicated GDI drawing object</td>
</tr>
<tr>
<td>SetBkColor()</td>
<td>Sets the background color of the text</td>
</tr>
<tr>
<td>SetPixel()</td>
<td>Draws a pixel according to the chosen color</td>
</tr>
<tr>
<td>SetTextColor()</td>
<td>Sets the color for the text</td>
</tr>
<tr>
<td>TextOut()</td>
<td>Displays a text message at the indicated coordinates</td>
</tr>
</tbody>
</table>
default, white is the color of the brush. This setting can be changed by modifying the parameters in the brush’s constructor, CBrush().

Text is displayed using the function TextOut(). By default, text is displayed using black color with a font of size of 12. These default settings can be changed by calling the appropriate functions. Color is created using the function RGB(). The color of the text and its background can be changed using the function SetTextColor() and SetBkColor(), respectively. For example, the following statements change the text color to green and the background to black:

```c++
dc.SetTextColor(RGB(0,255,0));
dc.SetBkColor(RGB(0,0,0));
```

### Numerical Functions

MFC does not have special numerical functions for performing scientific computations, as the functions in the standard C++ library are sufficient for most applications. Some of the most common functions in the C++ library are listed and described in Table 1.4. These functions are available for use in an application by inserting their prototype file, math.h, in the preprocessing area. In using the functions, care must be taken in considering their domain and range correctly. For example, \( \log(-2) \) will result in a crude answer as this function supports only positive numbers in its argument.

### 1.3 WRITING APPLICATIONS USING MFC

MFC provides powerful support for creating desktop or Web applications. This feature is observed through the rich features in MFC that allow the application to include a lot of advanced routines. Applications using Microsoft Visual C++ can be developed either on a guided or nonguided basis. In a guided approach, a tool called a wizard is provided to help in writing the code for the application. The wizard provides the initial skeleton of the program, and, therefore, the programmer does not have to worry about the detail steps in Windows such as calling functions for serializing objects and registering the application on the Windows interface. The programmer can concentrate on writing the code for the application. Many tedious jobs, ranging from a simple task like declaring a variable to something more difficult like creating a dialog box for an application, are done using the friendly menus provided by the wizard. Programming looks easier and more appealing this way.

However, a full guided tour has its drawbacks. The programmer may not benefit too much from the “free ride.” It is important for us to know how to walk the stairs instead of using the elevator all the time. Climbing stairs is a generic skill every human being must possess. Taking a ride on the elevator is a luxury in the sense that elevators may not be available in many places. Therefore, persons lacking in this generic skill may not survive under certain conditions. The person may also not be
flexible enough to exercise several different options for meeting new challenges. Many fundamental steps such as creating a variable and declaring a class are considered the basic attributes of a language that a programmer should know.

Our approach in this book will be mostly to use the nonwizard option as we would like to concentrate on discussing the problems by writing small programs and keeping code writing to the minimum. The wizard approach involves some massive handling of the dialog windows and menus, which generate many files that are not related to the applications. A discussion on wizards in Chapter 3 should provide some relationship between the two options.

Creating a New Project

We start by discussing some basic ideas in creating an application on Windows. Microsoft Visual C++.Net provides an interface called Visual Studio for developing an

| Table 1.4 Some of the most common numerical functions available in the standard C++ library |
|----------------------------------|---------------------------------------------------------------|
| Function                        | Description                                                                 |
| double exp(double x)            | Returns the exponent value of its argument. For example, exp(-1) returns 0.3679. |
| double log(double x)            | Returns the logarithm value of its argument. For example, log(4.5) returns 0.6532. |
| double sin(double x)            | Returns the sine value of its argument. For example, sin(4.5) returns -0.9775. |
| double cos(double x)            | Returns the cosine value of its argument. For example, cos(4.5) returns -0.2108. |
| double tan(double x)            | Returns the tangent value of its argument. For example, tan(4.5) returns 4.6373. |
| double sinh(double x)           | Returns the hyperbolic sine value of its argument. For example, sinh(-1) returns -1.1752. |
| double cosh(double x)           | Returns the hyperbolic cosine value of its argument. For example, cosh(-1) returns 1.5431. |
| double tanh(double x)           | Returns the hyperbolic tangent value of its argument. For example, tanh(-1) returns -0.7616. |
| double asin(double x)           | Returns the arc sine value of its argument. For example, asin(0.4) returns 0.4115. |
| double acos(double x)           | Returns the arc cosine value of its argument. For example, acos(0.4) returns 1.1592. |
| double atan(double x)           | Returns the arc tangent value of its argument. For example, atan(0.4) returns 0.3805. |
| int abs(int x)                  | Returns the absolute value of its integer argument. For example, abs(4) and abs(-4) both return 4. |
| double fabs(double x)           | Returns the absolute value of its double argument. For example, fabs(4.5) and fabs(-4.5) both return 4.5. |
application. Besides C++, this interface is shared by other languages in the family including Visual Basic and Visual C#. In order to develop an application using MFC, a person must know the C++ language very well. A good knowledge of C++ is a prerequisite to developing applications on Windows. This is necessary since MFC has classes and objects defined in a manner that can only be understood if one knows the language well.

A C++ project can be created in many ways, depending on user requirements. Table 1.5 lists some of the most common ways to create an application with Visual Studio. In its simplest form, a standard C++ project that runs without the support of any Windows functions is a console application. This option is necessary to a beginner in C++ or a person who does not wish to use the Windows facilities. The console option is available by choosing New Project, Win32 Application and by choosing Console Application in Application Type.

A Win32 Project is an option for creating an empty application with or without the support of MFC. This option does not provide a guide for creating an application, as the person must know all the details. One advantage to this option is the small amount of code required to generate an application. The option allows the application to exist as an executable file (EXE) or as a dynamic-link library (DLL).

The MFC Application option is a guided approach for creating an application using a tool known as the wizard. With this option, the details about Windows are prepared by Visual Studio through a series of menus and dialog windows in the wizard. Therefore, the user can concentrate on writing the code for an application. The wizard does not provide the whole solution for the application as it only assists by generating the code related to the Windows management.

The Managed Extension option is a structured way of writing an application. This new option provides an opportunity to integrate the application with .Net frameworks such as Passport, .Net My Services, Windows XP, and Tablet PC.

Creating a Window

The easiest way to create a nonwizard application using Windows is to use CFrameWnd as the framework. CFrameWnd is a class derived from CWnd. CFrameWnd also has rich ancestry in other classes such as CCmdTarget, CObject, and CWnd, which allows access to many functions and variables for creating applications. In a nonwizard application, a window is created by deriving the class from CFrameWnd using the function Create() (see Table 1.6).

Table 1.5 Some of the available new project options

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Console Application</td>
<td>Native C++ project that supports no Windows</td>
</tr>
<tr>
<td>Win32 Project</td>
<td>Empty project with or without MFC</td>
</tr>
<tr>
<td>MFC Application</td>
<td>Wizard approach to creating a Windows application</td>
</tr>
<tr>
<td>Managed C++ Application</td>
<td>Managed C++ project with or without Windows support</td>
</tr>
</tbody>
</table>
The main window is created using the function `Create()` from the `CFrameWnd` class. In its simplest form, the main window is created as follows:

```cpp
Create(NULL, "My Main Window")
```

Create() has several parameters but only the first two need to be stated, as shown above. The first parameter indicates the default class used, whereas the second is the title of the application. Leaving the other parameters as is means we agree with the default settings of the window.

Several child windows are also created using the function `Create()`. A child window is a window derived from the main window. A child window can exist in the form of a push button, a list view window, an edit box, or a full window similar to the main window. To create a child window, an object must be derived from its class. For example, to create a push-button window on the main window an object is derived from the class `CButton`, and applied as follows:

```cpp
CButton MyPushbutton;
MyPushbutton.Create("My Button", WS_CHILD | WS_VISIBLE |
    BS_DEFPUSH_BUTTON, CRect(30, 325, 250, 355), this,
    IDC_MYBUTTON);
```

The above statements create an object called `MyPushbutton` derived from the class `CButton`. We will discuss the parameters inside `Create()` in later chapters. The `MyPushbutton` object is a push-button window shown as a three-dimensional rectangle in the main window. Several other types of child windows are created and displayed in a similar manner.

A Windows application consists of a client/server process represented as objects. The server occupies the main window using the functions `OnPaint()` or `OnDraw()`. These two special functions are the message handling functions that respond to the event detected by WM_PAINT. The device context object for these functions are created from the class `CPaintDC`. A client area is created using any function other than `OnPaint()`. In this function, a device context object is created by deriving its object from the class `CClientDC`.

## 1.4 WRITING THE FIRST NONWIZARD APPLICATION

In this section, we discuss a nonwizard approach for creating a simple application. The option can be started by choosing the appropriate icons and answering a series of questions, as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create()</td>
<td>Creates the main window for the application</td>
</tr>
</tbody>
</table>
Step 1: Start Visual C++. From the menu choose File, followed by New and Project, as shown in Figure 1.2. This step creates a new project.

Step 2: The screen, as shown in Figure 1.3, appears. Click the Win32 Project icon, for nonwizard MFC applications. Name the filename Code1 and choose a suitable folder for storing the project. Click OK to confirm the selection.

Step 3: The dialog window in Figure 1.4 appears. Choose Application Settings to set up the nonwizard features into the application. Choose Windows Application and Empty Project for developing an application using the nonwizard Windows option. Click the Finish button to complete the selection.

Step 4: The menu as shown in Figure 1.5 appears. Choose Properties to embed the features into the application.

Step 5: The dialog window as shown in Figure 1.6 appears. Highlight Use of MFC and choose the Use MFC in a Static Library option. This option embeds MFC into the application to produce an EXE file.

Step 6: Highlight Source Files in the Solution Explorer and right-click. The menu as shown in Figure 1.7 appears. Choose Add then Add New Item to insert two new files into the project.

Figure 1.2  Creating a new project.
**Figure 1.3** Win32 project option for MFC applications.

**Figure 1.4** Specifying an empty Windows application.
Figure 1.5  Specifying the properties in the application.

Figure 1.6  Specifying the static MFC library.
Step 7: A dialog window as shown in Figure 1.8 appears. Choose Header File and name the new file as Code1.h.

Type the following code into Code1.h:

```cpp
// Code1.h
#include <afxwin.h>

class CCode1 : public CFrameWnd
{
public:
    CCode1();
    ~CCode1() {}
    afx_msg void OnPaint();
    DECLARE_MESSAGE_MAP();
};

class CMyAppClass : public CWinApp
{
public:
    virtual BOOL InitInstance();
};
```

Step 8: Repeat Steps 6 and 7 to add the file Code1.cpp. Insert the following code:

```cpp
// Code1.cpp
#include "Code1.h"
```
CMyAppClass MyApplication;
BOOL CMyAppClass::InitInstance()
{
    CCode1* pFrame = new CCode1;
    m_pMainWnd = pFrame;
    pFrame->ShowWindow(SW_SHOW);
    pFrame->UpdateWindow();
    return TRUE;
}

BEGIN_MESSAGE_MAP(CCode1, CFrameWnd)
ON_WM_PAINT()
END_MESSAGE_MAP()

CCode1::CCode1()
{
    Create(NULL,"My First MFC Program");
}