Teaching Visual C++ for C Programmers in Industry

Masaaki Mizuno¹ and Shimbu Yamagata²

¹ Department of Computing and Information Sciences, Kansas State University, Manhattan, Kansas, masaaki@cis.ksu.edu
² Hitachi Ltd., Hitachi Institute of Technology, 2-2 Kagahara, Tsuzuki-ku, Yokohama, Japan, syamagata@sy.itech.hitachi.co.jp

Abstract

Microsoft Visual C++ (VC++) is an important development system for window-based applications. However, VC++ is known to be very difficult to learn. At Hitachi Institute of Technology, we have been offering a five-day intensive course on VC++ to Hitachi employees for the last 4 years. The prerequisite of the course is reasonably good knowledge and experience with the C language. Based on our experience in teaching the course, we have discovered difficulties that most programmers commonly share in learning VC++ and developed an effective approach to teach VC++. The course evaluation indicates that a majority of the attendants have been satisfied with the course and reported the improvement in their mastery of VC++ after taking the course. In this paper, we discuss the difficulties in learning VC++, report our approach, and conclude with evaluation by the course attendants.

1 Introduction

Microsoft (MS) Visual C++ (VC++) encompasses a C/C++ compiler, Microsoft Foundation Classes (MFC) library, and an Integrated Development Environment (IDE). VC++ is the most important development system for C++ based window applications. However, VC++ is considered to be a very difficult system to learn; its learning curve is reported to be at least 6 months for proficient C programmers [4].

Since the demand for quality VC++ programmers is high at Hitachi, we at Hitachi Institute of Technology (HIT) have been offering a five-day intensive course on VC++ to Hitachi employees. One of the authors of this paper has been assigned to teach the course for the last four years, and another author supervises the course and analyzes course evaluation forms completed by course attendants. We refer to the former as “the instructor” and the later as “the supervisor” in this paper.
We think that the difficulties of VC++ result from the following two factors:

1. The difficulty of the C++ language, particularly the object oriented aspects of the language, such as the notion of classes, inheritance, polymorphism, and class variables. The same claim is also found in [2].

2. VC++’s black-box nature of both the application development process and the application framework (AFX). AFX is a set of objects constructed from MFC classes that provides an execution environment for an application program. A programmer heavily uses two helper tools, called AppWizard and ClassWizard, to develop a window application. AppWizard generates AFX, and ClassWizard generates essential code for message routing, accessing control objects, etc. These tools are very powerful and successfully hide tedious tasks in programming from programmers. However, because of this successful hiding, novice programmers often fail to grasp the whole picture of the program development process and the run-time behavior of the window application.

The above observation is shared among attendants who tried to study VC++ through books before attending the course. Many VC++ books on the market teach VC++ programming by going through hands-on exercise problems. Interestingly, their comments can be almost unanimously characterized by the following three points:

1. They had difficulties in following C++ solution code in the books.

2. By following step-by-step instructions in hands-on exercises, they could develop programs. However, when they tried to develop their own programs even slightly different from the exercise programs, they did not know where to start.

3. Even though they managed to create their own programs, they could not debug the programs once they encountered bugs.

The last two comments exactly reflect the instructor's own experience when he studied MFC for the first time. After reading several books, the instructor gave up studying VC++ through hands-on exercises in the books. Instead, he traced the execution within AFX by using the VC++ debugger and also read several MFC source code files (in particular, macros) to understand the internal behavior of AFX. After such efforts, the instructor could clearly understand the meaning of each step of once-given-up hands-on exercises in the books (that is, each step of interactions with AppWizard and ClassWizard and the programming development process). The instructor later found that this approach of learning MFC was compatible with the contents of the book [5]. The book explains implementation details of the MFC, and we have learned that it is a popular book among MFC professionals.

This approach seems to contradict the claimed “advantage of OO programming;” that is, objects may be used as parts or building blocks to construct a system only by knowing their interface specifications and without knowing implementation details or internal representation of them. However, we claim it is not. Based on our experience from both learning and teaching, we emphasize that it is not possible to construct a software system by leaving everything completely in a black-box. It is essential to understand the structure and behavior of the target software
system and how parts (objects) are glued together in the system (i.e., how objects interact with the framework). However, once we understand them, we can use each part (object) without knowing the implementation details of it.

In this paper, we identify the difficulties in learning VC++ and report our approach to teach VC++ effectively in a five-day intensive course. We believe that even in a short course like ours, it is important to spend enough time on explaining core C++ language features to give the attendants clear understanding of the language before starting discussion on MFC. We also think that understanding the internal behavior of AFX and window applications is important to master VC++. In particular, showing real uses of the C++ language features in MFC is effective in understanding both the language itself and the behavior of MFC. Note that many of the discussions in this paper also apply to Java and Java Foundation Classes (JFC), and we hope the paper helps those who want to learn Java/JFC as well.

The rest of the paper is organized as follows: In Section 2, we will briefly introduce Visual Basic (VB) and the development process of VB applications. VB is the most popular development system for window applications and is considered to be much easier to learn than VC++. We do this to clarify the difficulties in learning VC++ by comparing, in the later sections, the development processes for window applications in both VB and VC++. Section 3 overviews the application development process with VC++. Section 4 is the main section of the paper and identifies the difficulties in learning VC++ and our approach to effectively teach VC++ in the course. Section 5 will summarize the course evaluation by the attendants. We conclude the paper in Section 6.

2 Window Application development in Visual Basic

Microsoft provides two major development languages/environments for window applications; VC++ and Visual Basic (VB). VB is considered to be a much easier language to learn than VC++. In this section, we will briefly overview the development process of VB and the structure of VB applications to clarify in the later sections what contribute to the difficulties of application development in VC++.

A VB application is form based. An application consists of a form with various control objects, such as text box, list box, radio buttons, and check box. A user communicates with the VB program through such controls. In the development process, a programmer first defines a form using the form editor, by placing various control objects and setting various properties of the control objects. Associated with a form, VB creates one file. VB stores in the file all properties of the control objects placed on the form. A VB run-time system (in either compiled form or interpreter form) displays the form based on the properties in the file. An application can have more than one form, and a user can move from one form to another at run time. In such an application, each form has an associated file.

The programmer then specifies handler functions for events to which the application program should respond, such as handlers for mouse clicks and keystrokes. The skeletons of such handler functions are created below the list of properties in the file associated with the form. The programmer completes those handler functions by writing their bodies. It is also possible to
create a file consisting of variables and helper functions that are not associated with any form. Event handler functions may call such helper functions.

We think that the ease of VB programming is a result of the following factors:
1. All events triggered on the form (including events on control objects placed on the form) are handled by functions defined in the associated file. This structure is easy to understand and intuitive, and most programmers do not bother even to think about the mechanism behind the scene on how the event handler is invoked at run time.

2. VB provides a flat address space. There are only three types of variable scopes: local variables which are local to a function that declared them, module variables which are local to the file that declared them, and global variables which can be accessed by functions in any file. System functions are located in the global address space. Event handler functions and helper functions may call any system function from any place.

3 Window application and application framework (AFX)

In this section, we overview the behavioral structure of the Windows Operating System (OS), a window application, and the application framework (AFX). We also give a brief overview of how a programmer develops a window application program in VC++. Window applications are like icebergs floating on the sea of the Windows OS (Figure 1). Each iceberg of a window application created by AppWizard of VC++ consists of AFX and the application code (the part of the window application that the programmer actually codes). The application code is located at the tip of the iceberg.

The Windows OS maintains the system message queue, and each window application maintains a window application message queue (Figure 2). Whenever a user moves the mouse, clicks the mouse button, or types on the keyboard, the device driver for the mouse or keyboard converts the input into a message and places it in the system message queue (thus, in the following description, we use “message” and “event” somewhat interchangeably). A Windows OS thread removes a message, one at a time, from the system message queue, examines it to determine the
destination window, and then places it in the message queue of the destination window application.

Every window application has two fundamental functions (or procedures, methods): the window's main function denoted WinMain and the window message routing function denoted WinProc. A thread, called a primary thread, is assigned to each window application, and executes WinMain. WinMain executes the initialization code and then enters a loop called the message loop. It continues to execute the message loop until the window application terminates. The primary thread executing in the message loop removes messages, one at a time, from its application message queue, and executes a system call DispatchMessage with each message. The DispatchMessage call causes a Windows OS thread to call WinProc of the window application with the message as an argument. WinProc locates the message handler function (or method)\(^1\) for the given message and executes the method\(^2\). Thus, a window application simply repeats the above process of picking up a message from its message queue and asking the Windows OS thread to call the corresponding message handler method via WinProc.

Simply stated, the job of a programmer is to write message handler methods, in the tip of the iceberg (i.e., in the application code), for events to which the application needs to respond. For example, if the window application needs to respond to a right mouse button click, the programmer needs to write a handler method for the right mouse button click event.

Now, let us introduce how a window application is developed in VC++. A programmer starts the development with a special helper program, called AppWizard, informing AppWizard of various options regarding structure and functionality of the window application, such as the choice between single document interface (SDI) and multiple document interface (MDI), the database support, compound document supports, and the basic functionality of the window (docking toolbars, 3D control, etc.). Based on the information, AppWizard constructs AFX and a set of almost empty classes for the application code. Let us call such empty classes skeleton classes in this paper. AFX consists of objects instantiated from MFC classes (with appropriate attribute values) that implement the fundamental functionality of the window application and the options that the user selected through AppWizard. Functions WinMain and WinProc are located in AFX.

The skeleton classes created by AppWizard include (1) the window application class, (2) the frame window class, (3) the view class, and (4) the document class. All of them are derived from (i.e., inherit) MFC classes, and therefore have their base-class functionality. The obligation of the programmer is to add necessary functionality to the skeleton classes by adding event handler methods. The programmer may also add other classes that are used by these handler methods. A helper program called ClassWizard is used to add handler methods. The programmer instructs ClassWizard as to which events the application will handle and in which classes. These events include mouse button clicking, key typing, radio button pushing, list box updating, and many

\(^1\) Since message handler functions are located in classes, we call them handler methods in this paper.
\(^2\) For messages of higher importance dequeued from the system message queue, the Windows OS thread directly calls the WinProc of the destination window application with the message, rather than placing the message in the application's message queue.
others. For each specified event, ClassWizard inserts the skeleton of the handler method in the specified skeleton class. ClassWizard also inserts a macro call that is used by AFX to invoke the handler method. Then, the programmer writes the body of the handler method.

4 Our approach to teach VC++

In this section, we will first describe the outline of the course, and then the subjects that the instructor emphasizes to enhance the understanding of the attendants.

4.1 Overview of the VC++ course

The course is a five-day course, 8 hours of class time per day. The course text consists of the following 7 chapters:
1. Overview of window application development with VC++
2. Classes, objects, inheritance, and polymorphism in C++
3. Form-based user interfaces (adding control objects on the form using the resource editor of VC++)
4. Adding event handlers for the user interface objects using ClassWizard.
5. Dialog Boxes
6. Graphic Device Interface (GDI) and non-form-based applications
7. Serialization

Each topic is covered by lecture that explains basic concepts of the subject and hands-on exercises that give practical experiences on the subject.

4.2 Class and inheritance structures of VC++ applications

VC++ is based on the C++ language. Therefore, unlike the flat structure of a VB application, an application program in VC++ is organized within the structure of the complex inheritance hierarchy of MFC classes. Consider a simple application that has only one form. Recall that VB creates only one file for such a simple application. The file will contain the attribute list of the form and event handler functions. On the other hand, in VC++ (assuming an SDI application with a form view), ClassWizard creates the following four skeleton classes for the programmer:
1. A class called a window application class which inherits the MFC class named CwinApp.
2. A class called a frame window class which inherits the MFC class named CframeWnd.
3. A class called a view class which inherits the MFC class named CFormView
4. A class called a document class which inherits the MFC class named Cdocument

These skeleton classes all inherit the functionality and attributes of their corresponding parent classes (base classes); those parent classes in turn inherit the functionality and attributes of their parent classes. Such relations go up all the way to the MFC class named CObject that is the ancestor of all the MFC classes. This mechanism provides a very rich and powerful programming environment. However, it is also very confusing for application programmers who only know non-OO languages, such as C. Furthermore, polymorphism, which is one of the key features of OO, is heavily used by AFX to invoke many methods in the skeleton classes.
Therefore, for C programmers, good understanding of class, inheritance, and polymorphism is essential. On the first day of the course, the instructor spends much time to explain class concepts, the inheritance mechanism, and polymorphism in detail. This way, as the course progresses, the instructor can explain how the AFX mechanism is implemented by these C++ features. At the same time, the attendants understand the importance of the OO features in C++ through real applications found in the AFX mechanism. This helps a great deal relieve the frustration caused by the black-box nature of AFX and VC++ application development.

4.3 Behavior of AFX

In a conventional terminal-based program, the programmer writes the flow of execution, starting with the main function, making some system calls, and terminating. On the other hand, in a window application, the flow of control is completely managed by AFX, as we have seen in the previous section. The execution (by the primary thread) begins with function WinMain. WinMain executes the initialization routine and then enters the message loop. Execution of the message loop causes a system thread to execute WinProc for each message, which searches the corresponding message handler method and executes the method. The programmer only writes event handler methods.

This programming paradigm is hard to grasp for console-based C programmers. After explaining the basic functionality of AppWizard and generating their first SDI and MDI skeleton programs using AppWizard in Chapter 1, we use the VC++ debugger to trace the execution of the primary thread, starting with WinMain and then executing the message loop. This is an effective approach to explain the structure and behavior of a window application and AFX.

4.4 ClassWizard and Message Routing

Using ClassWizard, the programmer adds event handler methods in the skeleton classes that are created by AppWizard. This process is confusing because each event may be handled by a certain set of skeleton classes and the set varies from event to event. For example, the handler for an event generated by manipulating a user interface objects, such as a menu item, a tool-bar button, and an accelerator key (for example, edit/cut menu-item), may be placed in any skeleton class (more technically, any class derived from MFC class CCmdTarget). On the other hand, the handler for an event caused by a left mouse button click may be placed in only the frame and view skeleton classes (more technically, classes that are derived from MFC class CWnd).

Another source of confusion is that C++ is a compiler-based language. The mechanism used by WinProc for message routing (that determines, for a given message, which handler method to invoke) can be seen by the programmer; that is, ClassWizard actually places macro calls for routing information in the source files of skeleton classes. Attendants who have C or C++ background are puzzled by these macro calls inserted by ClassWizard.

Compare those features with VB. First, there is no concept of skeleton classes for handler functions in VB (even though a programmer can define their own classes in VB), and any handler function for an event triggered on a form is placed in the file associated with the form.
Secondly, VB was originally an interpreter-based language, and its routing mechanism is completely hidden from the programmer. The recent versions of VB generate compiled code. However, the generated code heavily relies on run-time service routines, and VB still hides the mechanism. This is one of the reasons why VB applications run slower than VC++ applications.

Another source of confusion involved in using ClassWizard is that its user interface is not very intuitive. Many attendants reported (and the instructor agrees on this point) that even after learning the basic usage of ClassWizard in books by following their step-by-step instructions, they did not really know what they had done and that they were lost when trying to develop their own programs even slightly different from the exercise programs found in the books. Furthermore, there are actually situations in which the programmer has to add macro calls and event handlers without using ClassWizard (the situations include when the programmer wants to write a handler which responds to multiple events).

For those reasons, we concluded that it was important to explain what exactly ClassWizard does to the program and how the message routing mechanism (implemented in WinProc) works in AFX. There are three types of messages: (1) window messages, (2) command messages (WM_COMMAND messages), and (3) control notification messages. Window messages are sent to particular windows and can be handled only by skeleton classes derived directly or indirectly from CWind. Command messages are initiated by user interface objects and can be handled by any class derived directly or indirectly from CCmdTarget (CWind is derived from CCmdTarget). There is a specific order in which the objects of such skeleton classes receive command messages. Control notification messages are sent from control object to only the window on which the control objects are placed. WinProc in AFX uses routing tables, called the message maps, to route messages to the window application. Any class derived from CCmdTarget can have a message map. All objects of one class share the message map associated with the class. Each macro call placed in a class by ClassWizard is expanded to form one entry in the message map of the class. In addition, there are some methods in the skeleton classes that are invoked via the C++ polymorphism mechanism (called virtual functions), not via message routing.

In the course, we explain the routing algorithm in WinProc for each type of message. Combined with the trace of WinMain described earlier, the attendants gain a good understanding of what is happening in AFX when a window application is running, and AFX is no longer a mere black-box to them. In the exercise for Chapter 4, the attendants use ClassWizard to add skeleton event handler methods and write code in the methods. Before the hands-on exercise in the chapter, the instructor gives detailed discussion about the WinProc function in AFX and macros placed in the source files by ClassWizard. In the exercise, the instructor asks the attendants to pay attention to the meaning of each step of their interaction with ClassWizard and what code ClassWizard actually inserts in the source code as a response. In particular, the instructor constantly asks whether a handler method being added is called via the message routing (in such a case, which type of message) or polymorphism. This greatly improves their understanding of the application development process in VC++.
4.5 Macros and class variables

In VC++, macros play very important roles. All macro definitions, as well as other source code of MFC, are open to VC++ programmers. As discussed above, we explain the meaning of macros to form message maps. We also explain macros used for dynamic object creation (called CRuntimeClass, DECLARE_DYNACREATE, and IMPLEMENT_DYNACREATE) and for serialization (called DECLARE_SERIAL and IMPLEMENT_SERIAL). These macros are related to serialization covered in Chapter 7. Many books explain the serialization mechanism wrong. By reading the macros, the attendants obtain correct understanding of the mechanism. This makes the obligation of the programmer clear when using the serialization mechanism.

All the above macros declare C++ class variables (static variables). Therefore, understanding C++ class variables is important. We explain C++ class variables right before we explain the macros for the message maps in Chapter 4.

5 Course Evaluation

The course is offered at HIT twice a year: once in May and again in November. The instructor has taught the course seven times since November 1998. A total of 161 attendants have taken the course as of November 2001. At the end of each course, the attendants fill out evaluation forms. There are 23 categories used to evaluate the course and the instructor, using a scale of 0 to 4 with 4 being the most satisfied and 0 being the least satisfied. In addition, attendants are encouraged to write comments. Almost all attendants write comments.

The average scores of the two most important categories are as follows

- Course Effectiveness: 3.85
- Instructor Effectiveness: 3.94

The comments are very consistent across the attendants and course offerings. The positive comments that many attendants have written are:

1. “It was good to learn important C++ constructs on the first day of the course.” “I understand the importance of C++ in studying MFC.”
2. “I understand the importance of learning the internal behavior of the system.” “As I understand the internal behavior of the system clearly, it is now much easier to debug a program.”
3. “I have learned material that is hard to find or that is not covered by most books.” These types of comments refer to the message routing in WinProc and macros that the instructor covers in Chapters 4 and 7.
4. “As the days go by, I was getting to understand the lecture better.” “I brought a book which I did not understand before. Everyday I was getting to understand the book better. Now I can easily follow the book.”
5. “In each step of the hands-on exercises, I knew what I was doing.”

These comments effectively reflect the goals of the course that we have set. Negative comments are:
1. “The lecture was too fast.” “Slow down.” Close to a half of the attendants wrote these types of comments. The instructor well realizes that the lecture is fast. However, since the instructor wants to cover each subject in detail, he cannot slow down the lecture speed. An encouraging aspect is that many of the attendants who wrote such comments also accepted the speed and wrote that they would rather take the current style of lecture than less course material with slower lecture speed.

2. “I wanted to have more time for the hands-on exercises.” This comment is related to the comments in 1. In order to cover the subjects described in Section 4 in five days, the instructor spends less time on hands-on exercises. On the other hand, many attendants commented that they were glad that the instructor spent more time on lecture and cut time for the exercises. This type of comment is related to the positive comments found in 3; that is, they felt the lecture valuable.

3. “The level of the course was too high, and I could not follow.” There have always been a few attendants who did not have appropriate backgrounds to take the course; some do not know the C language, and others do not have enough maturity in their programming skills. For those attendants, the course was too difficult, and some of them had given up to follow the course. The instructor intentionally sets the level of the course high so that those who need to use VC++ at work can gain the most out of the course. We may need to consider a way to keep the attention of those who do not have appropriate backgrounds.

4. “There are too many classes and methods in MFC, and I still do not feel I can use VC++ at work.” The learning curve of VC++ is said to be at least 6 months for proficient C programmers. We recognize that there is a limitation as to what we can do in five days. However, after taking the course, the attendants understand the internal structure and behavior of window applications and become able to read books and documentation easier. We hope they will be able to study MFC classes and methods that were not covered in the course by themselves through books and documentation.

Even though there are some negative comments, we, as the instructor and the supervisor of the course, are in general satisfied with the responses by the attendants.

6 Conclusion

Microsoft VC++ is an important development system for window applications. However, VC++ is considered to be a very difficult system to learn. At Hitachi Institute of Technology (HIT), we have been teaching VC++ in a five-day intensive course. In this paper, we have identified the difficulties in learning VC++ that many C programmers share and described our approach to teach VC++ effectively. We attribute one of the difficulties in learning VC++ to the black-box nature of the application development process and the application framework (AFX). We think that it is important to understand the structure and the internal behavior of AFX. To understand the internals of AFX and interaction among objects forming AFX and the application code, it is essential to understand the OO features of the C++ language; namely, the concepts of objects, classes, inheritance, and polymorphism.
Object Orientation is known to be an effective methodology to develop large software systems. Encapsulation of data and methods within objects is essential to develop reliable systems. The effectiveness of inheritance and polymorphism is found in MFC and JFC. Unfortunately, these important features of OO are also the major obstacle in learning VC++ (and Object-Orientated programming). However, once a VC++ learner passes this obstacle, he will benefit much from many benefits that OO methodology provides.

We think another advantage of OO is that software development in OO requires careful analysis and design before implementation. Much research has been done in the areas of OO analysis and design. The most recent development in the areas is Rational Unified Process (RUP) [3] in Unified Modeling Language (UML)[1]. RUP in UML is considered to become the de facto standard in the OO development methodology. With RUP, well-structured OO programs can be effectively developed. We hope that the VC++ course at HIT helps our programmers enter the rich world of Object-Orientation, not to mention mastering VC++.

References


Biographical Information

MASAAKI MIZUNO is a Professor in the Department of Computing and Information Sciences at Kansas State University. His research interests include operating systems and distributed systems.

SHIMBU YAMAGATA is a manager at Hitachi Institute of Technology. His research interests include information systems and computer/human interaction.