Foreword

The world of computing has always had one corner stone of particular interest to many, from educators to practitioners: languages. And programming languages in particular. Over the years, we have seen new languages come—and, much less frequently, old languages go. It is always tempting to focus on “the one” language of fashion of the day. In this very readable and instructive textbook, Stan Warford has done the unusual—and risky—by taking the programming language Component Pascal that is far from mainstream, although it does have roots that are among the strongest in the field.

Given that the concept of formal language, whether at the level of architecture, design, or implementation language, is central to our discipline, it is important that students continue to be exposed to a wide variety of languages. No single language does everything perfectly, or even well, and students need to understand this fundamental tradeoff. The same holds for frameworks and programming models that need to be designed to allow harmony between the natural ways of a language and the needs to a framework for a particular domain.

I had the privilege of being one of the key designers of this language, together with my friends at Oberon microsystems. One thing we knew back the early 1990s, when we started this effort, was that a language alone wasn’t any good anymore. So we co-designed the language Component Pascal and the environment BlackBox Component Builder, which unites an application framework, a development environment, and an application runtime environment—in many ways bringing together the advantages of highly dynamic languages and systems like Lisp or Smalltalk and those of statically checked languages like Modula-2, Oberon, Java, and now C#.

Stan succeeds in providing a highly original introduction to programming. One that alternates between the high-level aspects of immediate user impact, such as dialogs and views, and the low-level concepts of core programming, such as loops and recursion. As a result, students are always able to anchor what they learn in application scenarios that make immediate sense. The ability to create functional programs quickly, while not deviating to adopt an arsenal of “dirty tricks”, will help keep students motivated. The clean underlying approach and overall structure will at the same time lead to a deep understanding of principles and rigor.

Clemens Szyperski
Redmond, July 2002

Editor's Note

What do you say after "Hello World"? Everyone who teaches programming has an answer. For Stan Warford, the question is meaningless, because in his book and lectures, he does not start with "Hello World" or even sequential IO. Rather, he starts with modules, interfaces, and the construction of dialog boxes. Yet his book is not one of those "click-here-and-then-click-there" lightweights, which merely describe the user interface of some currently popular development system. He sticks to all the classical topics—like structured algorithms, searching and sorting, stacks, lists, and trees—which should be part of any serious course in computing. However, he does not do this job in an old-fashioned way but with a unique approach introducing all the modern things we did not have ten years ago (at least not in introductory books): GUIs, components, frameworks, UML, design by contract, design patterns. He gets all this together with solid theoretical basics like grammars, EBNF, verification, GCL, complexity. He explains every notion and every line of his example programs thoroughly. The text—simple, clear sentences—is accompanied by numerous figures drawn carefully, so that every student will easily understand even complicated facts.

These are only a few reasons why I like Stan's book. I liked it since I first saw a preliminary version. At that time I was working on my own book "Module, Klassen, Verträge", which is the first German language book using Component Pascal and BlackBox. I even considered abandoning my book, because what could I say that Stan did not already say in his? Fortunately, with the help of my colleague Helmut Ketz at Reutlingen University, I found an approach for my book that complements Stan's approach. Then I got a chance to help publish this book, for which I am most grateful. Thank you Cuno Pfister and Wolfgang Weck at Oberon microsystems and Reinald Klockenbusch at Vieweg!

Now this is the first English language book using Component Pascal and BlackBox. Since both our books fit together well, we have prepared an online service for the readers of both books, which is available at

`http://userserv.fh-reutlingen.de/~hug`

It includes a glossary in English and German covering topics of both books. May it serve you well!

Karlheinz Hug
Reutlingen, September 2002

Preface

This book is the outgrowth of an introductory computer science course taught at Pepperdine University. The course, as well as the book, is primarily for majors in computer science who intend a more in-depth study later, and secondarily for non-majors who desire a strong background in computers so they can deal with them effectively in their chosen fields.

For years we had followed the common practice of basing the course on the programming language Pascal. The emergence of many ideas in the field of software engineering led us to reevaluate the goals of the introductory course and how they could be effectively achieved. Rather than to simply append the new ideas onto the old approach we found that a paradigm shift was necessary, which had the effect of changing both the content and the organization of the course. Those ideas as reflected in this book include:

The ideas behind this book

- Frameworks
- Graphical user interfaces
- Object-oriented languages

There is an interrelationship between all these ideas. Each idea gains a measure of force in conjunction with the others, so that the whole is more than the sum of its parts.

Frameworks

A framework consists of a complete programming environment including a text editor and all the other tools needed to write and execute programs. The BlackBox framework is an exceptionally powerful but simple framework for software development. Power and especially simplicity are desirable attributes for teaching introductory programming. Like all good frameworks, BlackBox is platform independent, so that students and teacher need not complete their work with the same operating system. Furthermore, the cost of the framework for academic use is free, with the complete development system available for downloading from the Internet.

A framework is more than a collection of code libraries that provide an application programming interface for the developer. Like many development environments, the BlackBox framework does not require the programmer to write the event loop for the graphical user interface (GUI). Unlike most systems, however, which generate the event loop with code skeletons for the programmer to fill in to handle interaction events, the BlackBox framework *is* the main program and the programmer's application is simply an extension of the framework. The event loop is hidden and there is no handwaving to explain confusing code skeletons. The benefit of this arrangement is that students learn distributed control from the outset, which is an important aspect of object-oriented programming.

Distributed control

Another advantage of the BlackBox framework for teaching programming principles is its use of design by contract, which is also prominent in the Eiffel system. The idea is that every service provided by a method or procedure has preconditions that must be satisfied by the client. If the preconditions are met the method or procedure is guaranteed to function correctly, satisfying the postcondition. The documentation of the framework reflects this philosophy by specifying preconditions and postconditions for its services. Furthermore, the ASSERT statement is a primitive in the programming language, so that students can specify their own pre- and postconditions in their programs. There is a correspondence between pre- and postconditions of the framework and the Hoare triple of formal methods. Our introductory sequence requires a concurrent course in formal methods and integrates those topics with the programming course based on this book. The formal methods material is segregated in optional starred sections for the benefit of those whose curriculum does not include this requirement.

Optional starred sections for formal methods material

Graphical user interfaces

Using an object-oriented framework in the first year caused a reassessment of the entire issue of input/output. As it turns out, there is a natural one-to-one replacement of old topics with new ones. In place of interactive I/O, where the program prompts the user for input with a command line interface, is dialog box I/O. In place of file I/O is window I/O.

The BlackBox framework provides a Forms subsystem, which enables the student to program a GUI. A form is a component that is implemented with the model-view-controller (MVC) design pattern to produce a dialog box. The beauty of the framework is that students can begin programming immediately with dialog boxes without having to know the details of the MVC design pattern. The services provided by the Forms subsystem are so simple and so powerful that the programs we formerly assigned using Pascal with interactive input are longer than the equivalent programs with a dialog box. Programming with a GUI from the outset in the introductory course motivates students, because they learn how to produce programs with a professional appearance.

Dialog boxes

The simplicity of using the Forms subsystem with BlackBox is in marked contrast to the equivalent task of programming with Java's AWT or Swing library of classes. Programming with the Forms subsystem requires no concept of listeners or of layout management. An input field of a dialog box is simply mapped by the framework to an exported variable. Consequently, the chapter on dialog boxes can immediately follow the chapter on variables.

In BlackBox, files take a secondary role to the GUI and are available at a lower level of abstraction than are windows. Although it is possible to perform file I/O in BlackBox this text replaces it with scanning from and writing to the focus window. The user achieves the corresponding file operation by selecting Open or Save from the File menu, which hides the mechanisms of file I/O. BlackBox provides a scanner

Window I/O

that is powerful for experienced programmers to use but proved too difficult to understand for beginners. This book uses a scanner designed for the introductory course that can scan integers, reals, characters, strings, one-dimensional arrays of integers and reals, and two-dimensional arrays of integers and reals. Window I/O requires slightly more understanding of the MVC design pattern and is introduced later than dialog boxes.

Object-oriented languages

There is a debate among computer science educators about the placement of object-oriented (OO) programming in the first year of instruction. Some argue that students should program with objects from the outset even to the exclusion of procedural programming, while others advocate a more gradual mixed approach. To a certain extent, the choice is dictated by the language and development environment chosen to teach the concepts. For example, if you teach Java, which attempts to be purely OO, in the introductory course then it is impossible to construct a function that is not also a method even if inheritance or polymorphism are not used. Languages like C++ or Ada are, by design, mixed languages. That is, they are not pure OO languages but have procedural features as well.

The BlackBox framework is based on a new language named Component Pascal. Because it is a derivative of Oberon/L, much evolution has occurred since Pascal was designed. The pedigree of the language is

Pascal → Modula → Oberon → Oberon-2 → Oberon/L → Component Pascal

Unlike C++ and Ada, which added object-orientation onto an existing procedural language, Component Pascal is designed with no backward-compatibility requirements. However, like C++ and Ada, Component Pascal is by design a mixed-paradigm language with procedural and OO features.

The BlackBox framework lends itself to a gradual, mixed approach in the exposition of software design. For example, the dialog boxes provided by the Forms subsystem require procedural programming even though the subsystem is itself based heavily on objects. The approach taken in this book is more evolutionary than revolutionary. Its goal is to move the student through successively higher levels of abstraction, starting with procedural programming and concluding with OO programming.

The goal of this book

Some computer science educators claim that it is harmful to teach procedural decomposition to beginning programmers. They maintain that all data structures should be objects and that a pure object-oriented (OO) language should be used. Otherwise, you force students into the dreaded paradigm shift experienced by seasoned professionals when they switch to object orientation. Whether the dreaded paradigm shift occurs depends on how late in the student's academic career the shift is required. When recursion was new, we taught it late, and it was a shift. Now we teach it the first year and it is simply another tool. Scheme advocates maintain that recursion, along with the functional paradigm, should be taught at the outset, though most others incorporate it as one of many tools.

Certainly, object orientation belongs in the first year so the dreaded paradigm shift will not occur. But whether inheritance and polymorphism (which are the tools of OO design) should be taught at the outset is debatable. This book has abstraction as a theme, and students progress from lower levels of abstraction (procedural) to higher (OO). This progression has the advantage of teaching the abstraction process, which is a more general concept than object orientation. OO programming is still taught in the first year, minimizing the paradigm shift. In my experience, students can learn procedural programming first without harm as long as they are not steeped in it for an extended period of time before progressing to OO.

Like most languages designed by Niklaus Wirth, Component Pascal is small, simple, elegant, yet powerful. Component Pascal has many features to recommend it:

- Modules—The module is the basic unit of compilation. Variables can be global to a module, which eliminates the necessity of the confusing static feature of C++ and Java.
- Interfaces—Unlike header files in C++, the interface is created automatically by the compiler. The framework automatically keeps track of consistency between compiled and recompiled modules.
- Memory protection—The C++ language provides explicit pointers but cannot insure against memory leaks. The Java language does not provide explicit pointers, but guarantees memory protection with its automatic garbage collection. Component Pascal is unique in that it provides both explicit pointers as well as automatic garbage collection.
- Object-oriented—Component Pascal is fully object-oriented with polymorphism and single inheritance.
- Dynamic linker/loader—The compiler generates fast native code. There is no virtual machine or byte code intermediate language. Modules are loaded on demand within the framework.

Unfortunately, Component Pascal also has one major weakness that prevents its widespread use as a language in the typical introductory Computer Science course. Namely, C++ is entrenched in the professional software development world, Java is entrenched in the Web world, and universities are pressured to produce graduates with specific skills in those two languages. The pressure in the introductory course comes internally from teachers of the upper-level courses who want their students to already have C++ and Java skills. Those teachers get the pressure externally from industry, which wants university graduates with those skills.

There was a time in computer science education when programming languages for the introductory course were determined not by industry pressure but by pedagogical and technical considerations. Whether those days are gone forever or will reappear after the supply of computer science graduates finally matches the extreme employment demand remains to be seen.

At Pepperdine University, students study Component Pascal during the first year—that is, two semesters—from this book. The second year they study C++ — again for two semesters. The third year they are introduced to the functional paradigm with Common Lisp, the declarative paradigm with Prolog, and the concurrent paradigm with Java. Java is the primary language of instruction throughout the third and fourth years. This curriculum provides the benefit of Component Pascal in the introductory course while at the same time giving students the skills demanded by industry.

Resources

One of the best features of BlackBox is that the complete development system is available from Oberon microsystems at

`http://www.oberon.ch/`

and is free for educational use. The on-line documentation contains the defining language report and a sequence of tutorials (although geared to the experienced programmer, not the typical introductory computer science student). The educational version has the full programming capability of the developer version. At the time of this writing, Oberon microsystems does not supply the current 1.4 version of BlackBox for Macintosh operating systems. However, an older Macintosh version is still available at Oberon microsystems' ftp server

`ftp://www.oberon.ch/BlackBox/Mac/REL132/BB132A.HQX`

Note that the above URL begins with `ftp://` instead of the usual `http://`. Support for the Macintosh development environment is no longer available, and some interfaces have been changed in the transition from the 1.3 family to 1.4. However, none of the programs in this book use any of the changed interfaces, and all programs work as described on both MSWindows and MacOS. Documents are fully exchangeable between installations of the two releases.

All the programs in this book, as well as a set of lecture slides in PDF format, are available electronically at

`ftp://ftp.pepperdine.edu/pub/compsci/comp-fund`

Also at this site is a paper presented at an ACM SIGCSE conference that gives a few more details of our experience using BlackBox in the first-year course.

Acknowledgments

The designers of the BlackBox framework deserve congratulations for the sheer technical excellence of their product. Many people contributed useful suggestions and corrections to this manuscript in its various stages of development including Bill Bunch, Leighton Cowart, Reinhard Dietrich, Peter Fogg, Stephan Gehring, Dominik Gruntz, Rick Miltimore, Bernhard Treutwein, Wolfgang Weck, Russ Yost, and Brennan Young, as well as countless students at Pepperdine University who endured previous beta versions. I especially thank two individuals who had a large influence on the content—Professor John Motil, whose constructive criticisms of early versions of the scanners in the PboxMappers module improved them considerably, and Professor Dung “Zung” Nguyen, who provided many of the ideas for the data structure specifications and the object-oriented design patterns. At Vieweg, Professor Karl-heinz Hug has been most helpful in the publication process.

Stan Warford
Malibu, July 2002

Contents