Alberto Pettorossi

Programming in C++

00173 Roma
via Raffaele Garofalo, 133 A–B
tel. (39) 06 93781065 telefax (39) 06 72678427

www.aracneeditrice.it
e–mail: info@aracneeditrice.it

ISBN 88–7999–323–2

I edizione: ottobre 2001
II edizione: ottobre 2002
I ristampa: dicembre 2003

Contents

Preface

We present a collection of C++ programs through which we intend to illustrate some important programming techniques such as: (i) iteration, (ii) recursion, and (iii) backtracking. These techniques are very powerful ways of finding solutions to problems. They can also be viewed as efficient realizations of the *generate-and-test* method for problem solving. This method consists in constructing a space with candidate solutions, called the *search space*, and then visiting this space with the aim of finding the desired solutions.

The *generate* part refers to the construction of the search space itself. It may be the case that the search space is fixed in advance and it is independent of the problem at hand. It may also be the case that the search space depends on some intermediate results obtained during the search itself. For efficiency reasons one wants to keep the search space as small as possible, without losing any desired solution. For efficiency reasons it is also important to avoid the visit of the parts of the search space where there are no desired solutions.

The *test* part refers to the computation which takes a candidate solution and checks whether or not it is a desired solution. This computation effort is often not very expensive.

If the search space is fixed in advance then the iteration technique can be applied with success. If the search space varies while its visit progresses, then recursion and backtracking can be more effective. We will illustrate these points through various examples, including the ones for computing dispositions, permutations, combinations, anagrams, and the solution of the n-queens problem. In these cases we visit the search space in the depth-first fashion by making use of stacks. We will also provide an example of the breadth-first visit of a tree-like search space by making use of queues.

The recursion technique is also illustrated via some classical searching and sorting algorithms. Sometimes recursion can be reduced to iteration in a simple way, like, for instance, in the case of the computation of the Fibonacci numbers or the Towers of Hanoi problem or the heapsort algorithm.

In these collection of programs we also expose the reader to the following important programming techniques.

1. The use of pointers and recursive procedures for the manipulation of tree-like data structures. In these cases we have constructed copies of the structures to be modified, for avoiding the undesirable side-effects which may be due to structure sharing through C++ pointers. These undesirable side-effects occur also in other programming languages such as Pascal. The formal differentiation program will provide an example of this technique.

2. The use of files for the input and the output of values. The program for constructing a spanning forest of a directed graph will provide an example of this technique.

3. The use of strings and their parsing according to a regular grammar or a context-free grammar. From the given string a so-called *parse tree* is generated and suitable compu-

tations are then performed while visiting that tree. The program for proving theorems in propositional calculus and the parsing programs will provide examples of this technique.

The C++ language has given us the opportunity of presenting some simple techniques for achieving: (i) parametricity, and (ii) modularity.

By *parametricity* we mean the ability of defining abstract data structures which are parametric w.r.t. the type of the components objects through the use of *templates*. For instance, we can define lists whose elements are of type `T`, and then instantiate `T` to be `int` or `bool`, according to our need of having lists of integers or lists of booleans. By using templates, the code for the head, tail, and cons functions has to be written once only. Thus, templates allow us to declare a parametric data type and to have a basic form of polymorphism at our disposal. Other features of object oriented programming are provided by the use of the `class` and `struct` instructions.

By *modularity* we mean the ability of writing and compiling different program modules separately, and including them in our programs as separate units. By doing this, we can break up a large program unit into smaller units, and we can avoid the need for rewriting or recompiling the same code several times. Modularity is based on the use of the `#include` instruction which works like a macro call, and it allows us to include into our program some program modules which are available in a library.

Modularity is based also on the distinction between: (i) *function declaration* and (ii) *function definition*. A function declaration is written in a so-called *header file*, usually with extension `.h`, while a function definition in written in another file, usually with extension `.cpp`. When we write a program module we need to know only the function declarations, while the corresponding function definitions may be unknown to us. Moreover, the function definitions can be changed independently of the program units which use the corresponding function declarations, and these changes should not affect the use of the functions made by other program units.

At the end the book we address the following problems related to Numerical Analysis: (i) the search of the zero of a given function, (ii) the integration of a given function in an interval, and (iii) the solution of a system of linear equations. These problems are solved by using classical methods given in the literature.

Some of the programs we have presented in this book, show features of C++ which may be not so desirable. Among them, we want to mention the following ones.

1. The type discipline is weak. The boolean values `true` and `false` can be used as the integers 1 and 0, respectively. We can safely write `1 + true`. The presence of `void*` allows the programmer to treat an object of one type as it were of a different type.

2. The array type and the string type can be realized via pointers, and in some contexts the type difference between array, string, and pointer may be irrelevant.

3. The templates are not really polymorphic. One can write a template for pairs of values of type `T` and then `T` can be `int` or `bool` (so that we have pairs of integers or pairs of booleans). However, `T` itself can be neither the type of pairs of integers nor the type of pairs of booleans.

4. The distinction between expressions and statements is not sufficiently sharp, in the sense that statements, as well as expressions, have values. This fact may confuse the non-expert programmer. It may also obscure the readability of the programs. For instance, `int k; int i=(k=48); cout << i;` produces the output `48`.

This book can be viewed as the C++ version of the companion book; A. Pettorossi, *Learning Pascal through examples*, Aracne, October 1993. ISBN 88-7999-056-0. Some restrictions imposed by the programming language Pascal which we experienced when writing the programs of that book, are now overcome using C++.

All programs in this book were compiled using C++ compiler version 2.95.3. They run under Linux Mandrake 2.2.15-4mdkfb on a 497 Mhz Pentium III machine.

In order to learn the C++ language the reader may refer to the book [2].

Acknowledgments

I would like to thank the many persons who taught me the central ideas of programming, and in particular, R. Burstall, E. W. Dijkstra, P. Ercoli, R. Kowalski, and J. Reynolds. My gratitude goes also to my colleagues and students of the University of Roma Tor Vergata. Many thanks to Nicholas Amadori and Fabio Fioravanti for their suggestions and help. The presence of Maurizio Proietti has been of great support and encouragement, as always during the last fifteen years.

Alberto Pettorossi
Department of Informatics, Systems, and Production
University of Roma Tor Vergata
Via del Politecnico 1, 00133 Roma, Italy
email: `adp@iasi.rm.cnr.it`
URL: `http://www.iasi.rm.cnr.it/~adp`

List of Programs and Input Files