

Object-Oriented Software in C++

Object-Oriented Software in C++

Michael A. Smith

Senior Lecturer

University of Brighton

UK

江苏工业学院图书馆
藏书章



CHAPMAN & HALL

London · Glasgow · New York · Tokyo · Melbourne · Madras

Published by Chapman & Hall, 2-6 Boundary Row,
London SE1 8HN, UK

Chapman & Hall, 2-6 Boundary Row, London SE1 8HN, UK

Blackie Academic & Professional, Wester Cleddens Road,
Bishopbriggs, Glasgow G64 2NZ, UK

Chapman & Hall Inc., One Penn Plaza, 41st Floor, New York
NY 10119, USA

Chapman & Hall Japan, Thomson Publishing Japan, Hirakawacho,
Nemoto Building, 6F, 1-7-11 Hirakawa-cho, Chiyoda-ku, Tokyo 102,
Japan

Chapman & Hall Australia, Thomas Nelson Australia, 102 Dodds
Street, South Melbourne, Victoria 3205, Australia

Chapman & Hall India, R. Seshadri, 32 Second Main Road, CIT East,
Madras 600 035, India

First edition 1993

Reprinted 1994

© 1993 Michael A. Smith

Typeset in 10/12 pt Times by the author using Word 5 on a
Macintosh Quadra.
Printed in Great Britain by The Alden Press, Oxford

ISBN 0 412 55380 5

SPARC is a registered trademark of SPARC International, Inc.
UNIX is a registered trade mark of UNIX Systems Laboratories, Inc.
Sun is a trademark of Sun Microsystems, Inc.

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the UK Copyright Designs and Patents Act, 1988, this publication may not be reproduced, stored, or transmitted, in any form or by any means, without the prior permission in writing of the publishers, or in the case of reprographic reproduction only in accordance with the terms of the licences issued by the Copyright Licensing Agency in the UK, or in accordance with the terms of licences issued by the appropriate Reproduction Rights Organization outside the UK. Enquiries concerning reproduction outside the terms stated here should be sent to the publishers at the London address printed on this page.

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

A catalogue record for this book is available from the British Library
Library of Congress Cataloging-in-Publication Data available

Printed on permanent acid-free text paper, manufactured in accordance with the proposed ANSI/NISO Z 39.48-199X and ANSI Z 39.48-1984

Preface

This book is aimed at programmers who wish to learn the object oriented language C++. A knowledge of C, the ancestral language of C++, is not a requirement, as the book assumes no previous knowledge of this language.

The first two chapters concentrate on the basic constructs in the C++ language. The book then moves on to discuss the object oriented features of the language, using numerous examples to illustrate the ideas of encapsulation, inheritance, and polymorphism. In illustrating these ideas, the discussion is initially restricted to the high level features of the language. Templates are introduced at an early stage to encourage users to write re-usable classes.

Once these more fundamental points have been explained, the book then looks at the low level features of the language, in particular address arithmetic. The introduction of address arithmetic and pointers is deliberately deferred until the later chapters of the book, in order to encourage users of the language only to employ these features using the class mechanism. Various classes using pointers have been included to illustrate the use of these features to build constructs that can easily and safely be used by a programmer. There follows a chapter on descriptors, explaining how to build efficient implementations of complex data structures.

Separate chapters are devoted to container objects and persistence of objects. The book concludes with chapters on the attributes of a C++ program, and a summary of the important constructs in the language.

Self assessment questions and exercises are suggested for the reader at various points throughout the book.

The book describes version 3 of the C++ language, the programs used to illustrate the language have been tested using a variety of compilers, including version 3.1 of the Borland compiler. Appendix 1 lists suggested changes to some of the programs to allow them to be run using version 3 of the AT&T compiler and version 7 of the Microsoft C++ compiler. The changes usually take the form of a 'work round' for a language feature not currently supported by these compilers.

Thanks to: Prof. Dan Simpson for encouragement and the loan of a quadra on which this book was produced, Brian Bailey, Corinna Lord, Dominic De Virto, Franco Civello, John English, Paul Taylor, Phil Siviter, Richard Mitchell, Sara English, BA4 and BSc2 1992/3 for many helpful suggestions and comments. In particular Corinna for putting up with long hours in the 'computer room' and many useful suggestions on presentation and style.

The source code for all the example programs used in this book, is available using anonymous FTP at the net address `unix.brighton.ac.uk` in the directory `pub/mas`. Alternatively, contact the author by email at the address given below with a request for the source code.

Michael A Smith
Brighton, April 1993

`mas@unix.brighton.ac.uk`

xii Preface

The example programs shown in this book follow the conventions:

Item in program	Example	Convention used
class member function	deposit	Is in lower case
class member variable	the_balance	Starts with 'the' and is in lower-case
class name	Account	Starts with an upper-case letter
const	MAX	Is in upper-case
enumeration	TRUE	Is in upper-case
macro name	NAME	Is in upper-case
parameter name	amount	Is in lower-case
typedef	Boolean	Starts with an upper-case letter
variable name	mine p_ch	Is in lower-case A pointer to an item will start with 'p_'

Glossary of terms used

ADT	Abstract Data Type. The separation of a data type into two components: <ul style="list-style-type: none">• the public operations allowed on instances of the type.• the private physical implementation of the type. (Data representation and the implementation of operations allowed on the data items)
C	A language originally designed by Dennis Ritchie used to rewrite the Unix operating system. C++ is almost a superset of this language.
Class	The specification of data items and the functions that are allowed to operate on these data items. A class allows the user the ability to define a new data type, with the functions in the class defining the operations that are allowed on instances of the new data type. A class can also be used to encapsulate functions and data items.
Compile time constant	An expression that the compiler can resolve to a constant during the compilation process. For example, '2+3*7' is a compile time constant, whereas 'cost + 10' is not.

Encapsulation	The grouping of data and the operations that may be performed on that data into a single unit that provides a limited view of the operations allowed on the data items.
Information Hiding	Allowing a user of an encapsulated item only a limited view of the items contained within the encapsulation.
Inheritance	The creation of a new class using the components from an existing class.
Instance	<p>The creation of a physical instance (object) of a data type. For example in the declaration:</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;">Account mine;</div> <p>mine is an instance of the type Account.</p>
Instantiation	<p>The creation of an object which deals with a specific type of item from a template class.</p> <div style="border: 1px solid black; padding: 5px; margin: 10px 0;">Safe_vec <int> vector;</div> <p>vector is an instantiation of the class Safe_vec <int>.</p>
Message	The name of an operation and any arguments required by the operation.
Method	The algorithm (code) inside an object that processes a message.
Object	An instance of a class.
Object Oriented	Using the concepts of objects, classes, inheritance and polymorphism.
Polymorphism	The ability to send a message to any object and have the object respond using its definition of the operation requested. For example, instances of the classes Diagram and Text would respond differently to the message display.
Type safe	The compiler verifying that the use of instances of a type in a program is appropriate.

Contents

Preface

1	Introduction - part 1	1
1.1	A first C++ program	1
1.2	A larger C++ program	3
1.3	Repetition: while	4
1.4	Selection: if	4
1.5	Other repetition constructs	5
1.6	Other selection constructs	6
1.7	Input and output	9
1.8	The , operator	11
1.9	Self-assessment	12
1.10	Exercises	13
2	Introduction - part 2	15
2.1	Introduction	15
2.2	Declarations of data items	15
2.3	Fundamental types of C++	16
2.4	Typedef	17
2.5	Const declarations	17
2.6	Enumerations	18
2.7	Arithmetic operators	18
2.8	Relational operators in C++	19
2.9	Logical operators	20
2.10	Bitwise operators	20
2.11	The sizeof operator	21
2.12	Promotion of variables	22
2.13	Casts	24
2.14	Shortcuts increment and decrement	25
2.15	Expressions	26
2.16	Summary of operators	26
2.17	Self-assessment	27
2.18	Exercises	28
3	Classes	29
3.1	Introduction	29
3.2	The class	30
3.3	Functions	32
3.4	Visibility of class members	34
3.5	Declaration of a class, together with an instance of the class	36
3.6	An electronic bank account	36
3.7	Self-assessment	41
3.8	Exercises	42

4	Separating interface from implementation	43
4.1	Encapsulation	43
4.2	Separate compilation of classes	45
4.3	Re-use	48
4.4	Self-assessment	49
4.5	Exercises	49
5	Functions	51
5.1	Introduction	51
5.2	Local variables	51
5.3	Returning a result to the operating system	52
5.4	Function prototype	52
5.4	Call by value/call by reference	53
5.5	Const parameters to a function	54
5.6	Recursion	55
5.7	Inline vs. out of line code	56
5.8	Overloading of functions	57
5.9	Different number of parameters	59
5.10	Default values to parameters	60
5.11	Matching a function call to a function declaration	61
5.12	Function templates	63
5.13	Order of function matching (overloaded functions)	65
5.14	Self-assessment	66
6	Arrays	67
6.1	Arrays	67
6.2	Use of arrays	68
6.3	Representation of arrays	73
6.4	Passing arrays as parameters to a function	73
6.5	Initializing arrays of objects	76
6.6	Case study: a histogram	76
6.7	A stack built using an array	80
6.8	Templates (building generic classes)	83
6.9	A computerized bank system	84
6.10	Self-assessment	88
6.11	Exercises	88
7	Static variables and functions	89
7.1	Static variables	89
7.2	Self-assessment	93
8	A case study using OOD	95
8.1	Four counters	95
8.2	Self-assessment	111
8.3	Exercises	111

9	Inheritance	113
9.1	A savings account	113
9.2	Call of a constructor in the base class	116
9.3	A saving account with tiered interest rates	117
9.4	Visibility of class members	120
9.5	Constructors and destructors	122
9.6	A class to describe a room	122
9.7	A class to describe an office	124
9.8	Multiple inheritance	125
9.9	Static binding	128
9.10	Inherited functions	128
9.11	Inheritance of the same base class	129
9.12	Self-assessment	131
10	Defining new operators	133
10.1	Defining operators in C++	133
10.2	The class Money	133
10.3	*this the current instance of a class	135
10.4	Declaration of instances of a class with an initial value(s)	135
10.5	Class constants	137
10.6	Use of friend functions	139
10.7	Conversion operators	142
10.8	Initializing arrays of objects	147
10.9	A string class	148
11	Polymorphism	153
11.1	Virtual functions	153
11.2	A bank account	156
11.3	An abstract class for a bank account	157
11.4	A derived interest-bearing account	160
11.5	A derived higher interest account	161
11.6	Advantages and disadvantages of polymorphism	167
12	Pointers	169
12.1	Introduction	169
12.2	Class component *this	171
12.3	Use of pointers in C++	171
12.4	From arrays to pointers	172
12.5	Pointers vs. arrays	174
12.6	Dynamic storage allocation	177
12.7	Use of dynamic storage	179
12.8	Structs	183
12.9	Dynamic vs. static storage allocation	184
12.10	Overloading the operators new and delete	184
12.11	Operators * and ->*	186
12.12	Pointers and polymorphism	188
12.13	Self-assessment	190
12.14	Exercises	190

viii Contents

13	Declarations	193
13.1	Storage declarations of derived types	193
13.2	Structures allocated	194
13.3	Function prototypes	195
13.4	Formal parameter declarations	196
13.5	Union	197
13.6	Bit-field	197
14	Safe arrays in C++	199
14.1	Introduction	199
14.2	A safe vector	200
14.3	A safe two-dimensional array	203
14.4	Associative arrays	207
15	Macros	213
15.1	Introduction	213
15.2	Source inclusion of files	213
15.3	Text substitution	213
15.4	Conditional compilation	215
15.5	Predefined names	218
15.6	Overuse of macros	218
15.7	Inclusion of header files	219
15.8	Parameter specification: variable number of parameters	220
15.9	Faking parameterized types	221
16	C style input/output in C++	225
16.1	Passing data to a C++ program	225
16.2	Access to C functions in a C++ program	226
16.3	Case study: a text file de-archiver	227
17	Descriptors	239
17.1	Descriptors	239
17.2	A number class implemented using descriptors	240
17.3	Set implemented by descriptor	245
17.4	Exercises	252
18	Containers and iterators	253
18.1	Containers	253
18.2	A class for a Bag	255
18.3	Specification and implementation	259
18.4	A class for an 'iterator for the bag'	260
18.5	Using different implementations of a bag	266
18.6	Self-assessment	268
18.7	Exercises	269
19	Persistence of objects	271
19.1	Introduction	271
19.2	Overview of process	271
19.3	The process	272
19.4	Making objects persistent: the class Float	274
19.5	Saving objects to disk	276
19.6	Exercises	278

20	Attributes	279
20.1	Introduction	279
20.2	Lifetime	279
20.3	Linkage	281
20.4	Scope	282
20.5	Visibility	283
20.6	Storage class	283
20.7	Modifiers	284
20.8	Type	285
20.9	Run-time execution of a program	286
21	C++: a summary	291
21.1	Declarations in C++	291
21.2	Array declaration	291
21.3	Enumeration declaration	291
21.4	Class declaration and implementation	291
21.5	Inheritance	292
21.6	Statement expressions	292
21.7	Compound statement	292
21.8	Selection statements	292
21.9	Looping statements	293
21.10	Arithmetic operators	293
21.11	Conditional expressions	293
21.12	Logical operators	294
21.13	Short cuts	294
21.14	Exits from loops	294
21.15	Skip to top of loop	294
21.16	Address operations	295
Appendices		297
Appendix A:	C++ style input/output	297
Appendix B:	C style input/output	302
Appendix C:	Useful functions	306
Appendix D:	Priority of operators	312
Appendix E:	String and character escape sequences	313
Appendix F:	Fundamental types	314
Appendix G:	Literals in C++	315
Appendix H:	Keywords in C++	316
Appendix I:	Compatibility of code	317
Appendix J:	References	318
Index		319

1 Introduction - part 1

This chapter looks at some very simple C++ programs. In introducing these programs the basic control structures of C++ are presented.

1.1 A first C++ program

Like most books on programming, this too starts off with an example program that writes a successful greeting to the user's terminal:

```
#include <iostream.h>
```

```
void main()
```

```
{
```

```
    cout << "Hello world" << "\n";
```

```
}
```

which would display the following message on a user's terminal when the program was run:

```
Hello world
```

In the above example program, { and } are used to bracket the body of the function main. This contains the expression `cout << "Hello world" << "\n";` which writes the string "Hello world" followed by a newline to the current output stream `cout`. This can be thought of as sending the messages "Hello world" and "\n" to the object `cout`. Normally `cout` would be 'attached' to the terminal. Figure 1.1 shows the structure of a C++ program.

Note: "\n" is simply the C++ way of expressing a string composed of the newline character. The \ character is used to specify that the next character has a special meaning, in this case newline. A full list of escape sequences is given in appendix E.

The line `#include <iostream.h>` is not part of the C++ language. It is a directive to the pre-processor to replace this line by the contents of the file `iostream.h`. This file contains definitions about the input output process. It is usually held in one of the system directories of the computer system. This line must always start in column 1.

2 Introduction - part 1

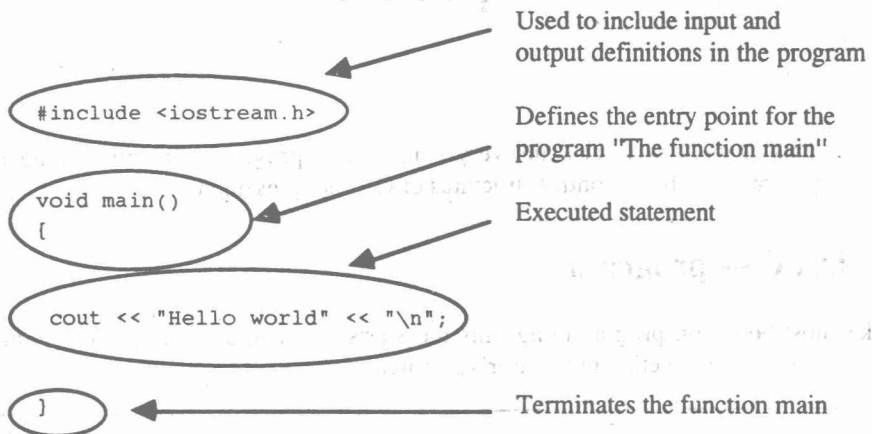


Figure 1.1 The structure of a C++ program.

The types of the items that are to be output may be mixed as in the case below. The C++ compiler uses the item's type to select the appropriate output form.

```
void main()
{
    cout << "The Sum of 1+2+3 is " << 1+2+3 << "\n";
}
```

Which would produce the following output when run:

```
The Sum of 1+2+3 is 6
```

1.1.1 Format of a C++ Program

A C++ program can be written without regard to format provided that the individual components that make up the program can be recognized. For example, the following is a valid C++ program:

```
#include <iostream.h>

void main(){cout<<"Hello world"<<"\n";}
```

Note: The directive `#include` must be on a line by itself and start in column 1. At least one white space character, for example space is required between any words that are alphabetic such as `void` and `main`, so that they can be individually distinguished.

1.1.2 Comments

C++ has two ways of introducing a comment into a program. Firstly:

```
/* An example comment */
```

Here the comment is bracketed between `/*` and `*/` although it is more usual to write this in the form:

```
/*
 * This program is a simple test of the C++ compiling system
 * and writes out the message Hello World to the terminal
 */
```

Note: The `/ */` comment delimiters may not be nested.*

Secondly:

```
// The rest of the line is a comment
```

Here the comment is introduced by `//` and is terminated by the newline.

Note: It is good programming practice to comment any code section that is not immediately obvious to a reader of the code.

1.2 A larger C++ program

A complete program to produce a 'count down' is shown below. In this program various constructs that affect the flow of control are introduced.

```
#include <iostream.h>

void main()
{
    int countdown=10;
    while ( countdown > 0 )
    {
        cout << countdown << "\n";
        if ( countdown == 3 )
        {
            cout << "Ignition" << "\n";
        }
        countdown--;
    }
    cout << "Blast Off" << "\n";
}
```

4 Introduction - part 1

When run this would produce:

```
10  
9  
8  
7  
6  
5  
4  
3  
Ignition  
2  
1  
Blast Off
```

1.3 Repetition: while

```
while ( countdown > 0 )  
{  
    ...  
}
```

The above statement repeatedly executes the code between { and } until the condition `countdown > 0` is no longer true.

Note: The ()s around the condition are mandatory.

The { and } brackets are only required if there is more than one statement to execute repeatedly. Many people, however, would always put in the {} to show the bounds of the loop.

1.4 Selection: if

```
if ( countdown == 3 )  
{  
    ...  
}
```

This executes the code between { and } if the condition `countdown == 3` is true.

Note: Equality is written ==

This can lead to many mistakes, as it is easily confused with assignment, which is written as =

A conditional expression will deliver 0 if false and 1 if true. As these are integer values, 0 may be used as false and 1 may be used as true.

In fact any value other than 0 is taken to be true, as in the case below:

```
if ( countdown )
    cout << "Not yet zero" << "\n";
```

Note: As only one statement was selected to be executed when the condition was true the enclosing { and } were not required.

1.4.1 if else

An else part may be added to an if statement as follows:

```
if ( countdown )
    cout << "Not yet zero" << "\n" ;
else
    cout << "Now zero" << "\n" ;
```

Note:

```
if ( countdown )
    cout << "Not yet zero" << "\n" ;
else
    cout << "Now zero" << "\n" ;
```

Must be included

The ; before the else must be present as it terminates the previous statement.

1.5 Other repetition constructs

1.5.1 for

The for statement in C++ is written as:

```
for ( int countdown = 10; countdown > 0; countdown-- )
{
    // ...
}
```

Note: The variable controlling the for loop 'countdown' may be declared inside the (/s).

6 Introduction - part 1

which in this example steps countdown through the values 10 to 1. This is equivalent to the following while statement:

```
int countdown = 10;
while ( countdown > 0 )
{
    ...
    countdown--;
}
```

*Note: `countdown--;` is the C++ idiom for `countdown = countdown - 1;`
In the for statement any of the components between the ;s may be omitted.*

1.5.2 do while

In some cases it is a requirement that the loop is executed at least once, in which case the do while statement may be used. For example, the above for statement could in this case have been written as:

```
int countdown = 10;
do
{
    ...
    countdown--;
} while ( countdown > 0 );
```

1.6 Other selection constructs

1.6.1 switch

The following rather inelegant series of if statements may be combined:

```
if ( number == 1 )
    cout << "One";
else if ( number == 2 )
    cout << "Two";
else if ( number == 3 )
    cout << "Three";
else
    cout << "Not One, Two or Three";
cout << "\n";
```