Object-Oriented Software in C++

Object-Oriented Software in C++

Michael A. Smith
Senior Lecturer
University of Brighton
UK

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

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

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 introduces 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 I 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 Vitto, 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 the 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

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: • 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	The creation of a physical instance (object) of a data type. For example in the declaration: `Account mine;` mine is an instance of the type Account.
Instantiation	The creation of an object which deals with a specific type of item from a template class. `Safe_vec <int> vector;` vector is an instantiation of the class `Safe_vec <int>`.
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

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.

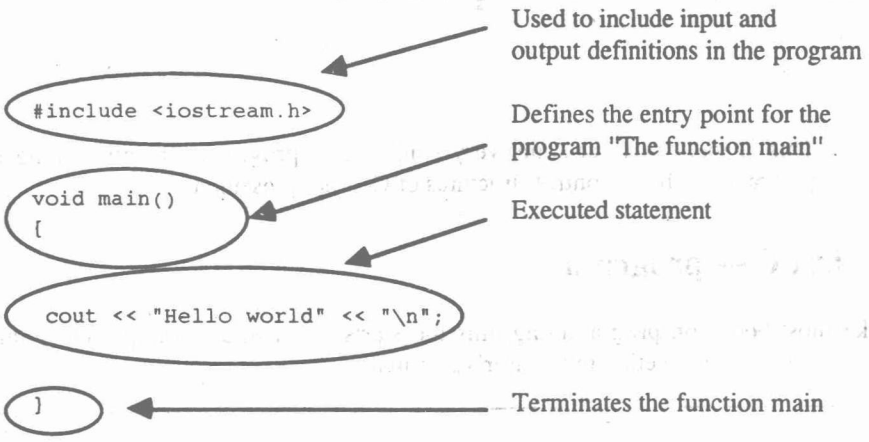


Figure 1.1 The structure of a C++ program.

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 delimeters 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";
}
```

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 )
{

}
```

The 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";
```

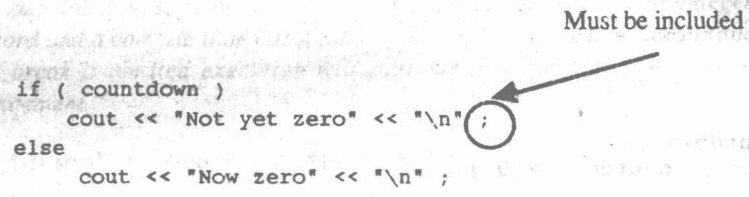
Note: As only one statement was selected to be executed when the condition was true the enclosing { and } were not required.

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:



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.

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";
```