

Preface

C++ is the most widely available and used object-oriented programming language. It is a very useful language that is successfully utilized by many programmers in many application areas. It is a reasonably carefully thought-out language where the design is based partly on acknowledged principles and partly on solid experience and feedback from actual use. The language is much more complicated and complex than, say, C and PASCAL. The key concepts of data abstraction and object-oriented programming are new to most people. One can use C++ effectively as a more strongly type checked C with a small amount of simple data abstraction and library use. This programming language has all the tools to write compact and comprehensive programs for finance, administration and statistics. Many of the most experienced individuals and organizations in the industry and academia use C++. Many programmers strongly prefer C++ over procedural languages such as C and PASCAL and deem the feature set provided by C++ as close to essential for their applications. C++ supports object-oriented programming. It supports it pretty well for real life applications. Its type system and general model of the world are coherent and relevant to real applications. C++ and its various implementations have a solid core where different compilers tend to agree and where the code generated is reliably good.

This book provides software solutions to problems in finance, administration and statistics. Most standard methods used in these fields are given with their programs in C++. In some cases the output of the programs is also displayed. Moreover the book gives an introduction to C++. Most of the programs are written in C++. Some of the programs are written in C.

All the standard methods such as number sorting, name sorting, applications of function templates, applications of class templates, inheritance, virtual classes, string manipulations are included.

In chapter 2 we give a some basic programs in C++. These programs are very helpful for the beginners. They are also the building blocks for more complex applications. Chapter 3 is devoted to the manipulation of strings. The class concept is introduced in chapter 4. Chapter 5 and 6 deals with function and class templates. collection of introductory program. Sorting and searching algorithms are implemented in C++ in chapter 7. Chapter 8 gives a collection of useful classes. They include string classes, vector classes, a matrix class, linked lists and trees. Chapter 9 is devoted to output/input streams. Here we consider C and C++ programs. In chapter 10 a collection of applications in finance are given. Applications in administration are considered in chapter 11. Problems in statistics are considered in chapter 12. Chapter 13 describes object-oriented analysis and design. A banking package is developed in chapter 14.

The level of presentation is such that one can study the subject early on in ones education in science. There is a balance between practical programming and the underlying language. The book is ideally suited for use in lectures on C, C++ and object-oriented programming. The beginner will also benefit from the book.

The reference list gives a collection of text books useful in the study of the computer language C++. There are a number of good text books for C++ available [2], [4], [5], [6], [7], [9], [12], [8]. We also refer to *The Annotated C++ Reference Manual* by Margret Ellis and Bjarne Stroustrup (1990)[4]. For data structures we refer to Budd (1994) [3]. For applications in science we refer to Steeb (1993) [10] and Steeb (1994) [11].

For a comprehensive discussion of the Boyer-Moore algorithm, the Floyd-Warshall algorithm, the sorting methods and the divide-and-conquer strategy we refer to Baase (1989) [1].

The C++ programs have been run under Borland C++ (version 4.5). The programs also run under Borland C++ (version 1.01) for OS/2 (version 2.1).

Without doubt, this book can be extended. If you have comments or suggestions, we would be pleased to have them. The email addresses of the authors are:

WHS@RAU3.RAU.AC.ZA
FS@RAU3.RAU.AC.ZA

DOS and Microsoft Windows are registered trademarks of Microsoft. Borland C++ and Turbo C++ are registered trademarks of Borland International. OS/2 is a registered trademark of IBM.

One of the authors (W.-H. Steeb) thanks his students from the Computational Science Programme of the National University of Singapore for stimulating discussions on object-oriented programming. They are: Fong Wei Leng, Heng Kok Liang, Tan Kiat Shi, Lee Wan Sie, Gan Chee Kwan, Toh Che Chou, Koh Mei Leng, Lee Yih, And Mieng Hwang, Teo Wee Chung, Lee Choon Meng. We also thank Catharine Thompson for her critical proof-reading of the manuscript.

0.1 Keywords in C++

C++ has the following keywords:

Table: C++ language keywords

<code>asm</code>	<code>double</code>	<code>new</code>	<code>switch</code>
<code>auto</code>	<code>else</code>	<code>operator</code>	<code>template</code>
<code>break</code>	<code>enum</code>	<code>private</code>	<code>this</code>
<code>case</code>	<code>extern</code>	<code>protected</code>	<code>throw</code>
<code>catch</code>	<code>float</code>	<code>public</code>	<code>try</code>
<code>char</code>	<code>for</code>	<code>register</code>	<code>typedef</code>
<code>class</code>	<code>friend</code>	<code>return</code>	<code>union</code>
<code>const</code>	<code>goto</code>	<code>short</code>	<code>unsigned</code>
<code>continue</code>	<code>if</code>	<code>signed</code>	<code>virtual</code>
<code>default</code>	<code>inline</code>	<code>sizeof</code>	<code>void</code>
<code>delete</code>	<code>int</code>	<code>static</code>	<code>volatile</code>
<code>do</code>	<code>long</code>	<code>struct</code>	<code>while</code>

Remark: `asm` is reserved for implementors to include a machine specific facility to pass information to an assembler. `extern` indicates that external linkage is to be made to this item. Obviously the placement of such an item must be known at compile time and hence must be implicitly static.

0.2 Fundamental Types of C++

The inbuilt data types or fundamental types are as follows:

Type	May also be specified as	Comment
char		Holds a character
unsigned		Unsigned version of char
signed char		Signed version of char
int	signed int signed	A machine word that holds an integer number
unsigned int	unsigned	Unsigned version of int
short	signed short int signed short	May have less precision than an int
unsigned short	unsigned short int	Unsigned version of short
long	long int	May have more precision than an int
unsigned long	unsigned long int	Unsigned version of long
long int	signed long int	
float		A number held in floating point form
double		May have more precision than a float
long double		May have more precision than a double

Note: A float, double and long double are stored as a floating point number. The rest of the fundamental types are stored as an exact quantity.

The unsigned types will be able to store a larger positive number than their signed types.

The precision of items declared with these data types may vary, depending on the compiler or machine used.

The representation of char may be either signed or unsigned. This is to allow implementors the choice of the architectural representation of a char.

As can be seen from the table above, there is no inbuilt data type for boolean values. C++ like C before, uses integer values to represent true and false.

Truth value	C++ delivers	C++ takes a truth value
true	1	Any value other than 0
false	0	0

0.3 C++ operators

The following table shows a summary of C++ operators in order of decreasing precedence.

For example

```
int u, x, y, z;
x = 3; y = 9; z = 2;
u = x + y % z;
```

firstly evaluates the remainder of

$$\frac{y}{z},$$

then adds this to `x` and assigns the result to `u`. Another example is

```
*g.value      (*g).value
```

The member selection operator `.` has higher precedence than the dereferencing operator `*`. Hence `*g.value` dereferences `g.value`, whereas `(*g).value` dereferences `g` and then selects the member value of the object to which `g` points.

The use of the operators given in following table will be illustrated with sample programs throughout this text. In C++ one can define operators for one's own data types. For example if one defines a data type for rational numbers, then one can define the addition, subtraction, multiplication etc. for this data type. This facility is called operator overloading. All operators can be overloaded (see discussion on operator overloading later in the text) except for

```
.      .*      ::      ?:
```

When overloading operators, one should bear in mind that the order of precedence and the syntax remains the same as that for the built-in data types.

The following table shows C++ operators in order of decreasing precedence. Each box holds operators of the same level of precedence.

Operator	operator name	example
::	scope resolution	<i>class_name::member</i>
::	global	<i>::name</i>
.	member selection	<i>object.name</i>
->	member selection	<i>pointer->member</i>
[]	subscription	<i>pointer[expr]</i>
()	function call	<i>expr(expr_list)</i>
()	value construction	<i>type(expr_list)</i>
sizeof	size of object	<i>sizeof expr</i>
sizeof	size of type	<i>sizeof(type)</i>
++	post or pre-increment	<i>lvalue++</i> or <i>++lvalue</i>
--	post or pre-decrement	<i>lvalue--</i> or <i>-- lvalue</i>
~	complement	<i>~expr</i>
!	not	<i>!expr</i>
, -+	unary plus and minus	<i>-expr</i> and <i>+expr</i>
&	address of	<i>&lvalue</i>
*	dereferencing	<i>*expr</i>
new	create (allocate memory)	<i>new type</i>
delete	deallocate pointer memory	<i>delete pointer</i>
delete[]	free array memory	<i>delete[] pointer</i>
()	cast (type conversion)	<i>(type) expr</i>
.*	member section	<i>object.*pointer-to-member</i>
->*	member section	<i>pointer->*pointer-to-member</i>
*, /, %	multiply, divide, modulo (remainder)	<i>expr / expr</i>
+, -	add, subtract	<i>expr + expr</i>
<<, >>	shift left, shift right	<i>expr << expr</i>
<, <=, >, >=	relational operators	<i>expr < expr</i>
==	equal	<i>expr == expr</i>
!=	not equal	<i>expr != expr</i>
&	bitwise AND	<i>expr & expr</i>
^	bitwise exclusive OR	<i>expr ^ expr</i>
	bitwise inclusive OR	<i>expr expr</i>
&&	logical AND	<i>expr && expr</i>
	logical inclusive OR	<i>expr expr</i>
? :	conditional expression	<i>expr ? expr : expr</i>
=	simple assignment	<i>lvalue = expr</i>
*, /=, =, -=, +=	multiply and assign, ...	<i>lvalue *= expr</i>
<<=, >>=	shift left (right) and assign	<i>lvalue <<= expr</i>
&=, =, ^=	AND (OR, XOR) and assign	<i>lvalue &= expr</i>
throw	throw exception	<i>throw expr</i>
,	comma (sequencing)	<i>expr, expr</i>