This chapter begins by describing the different kinds of Fundamental Data Type (FDT) in C++. The chapter then introduces identifiers (constant and variable) and the declaration and definition of identifiers. This is followed by discussing assignment; initialisation of identifiers, expressions and statements; operators; and type conversions. Basic input from the keyboard and output to the screen is covered, along with the use of manipulators for formatting input and output. The topic of casting (the process of converting an object of one data type to an object of another data type) is discussed. The chapter concludes by examining the auto, extern, register and static storage class specifiers for specifying exactly how a variable is to be stored, and the asm declaration for directly integrating assembly language code into C++ code.

4.1 Fundamental Data Types

There are four fundamental, basic or primitive data types in C++:

- **char**: Character: letter, symbol, punctuation and digit
- **int**: Integer: whole number
- **float**: Floating: point number
- **double**: Floating: point number

**char**, **int**, **float** and **double** are keywords or, more specifically, type specifiers. Each data type has its own attribute list which defines the characteristics of the type and may vary from one machine system to another. The types **char**, **int** and **double** each have variations, or data type modifiers, such as **short**, **long**, **signed** and **unsigned**, to enable more efficient use of the data types.

An additional character type is **wchar_t**:

- **wchar_t**: Wide character constant
There are also two additional types in C++ that I feel warrant mention at this point, namely \texttt{enum} and \texttt{bool}:

\begin{verbatim}
enum          Enumeration constant
bool          False–True enumeration
\end{verbatim}

Compared with the fundamental types, \texttt{enum} and \texttt{bool} may appear to be oddballs, but just bear them in mind at this stage until we discuss them in more detail at a later date.

The characteristics of each data type will now be discussed separately.

### 4.1.1 The \texttt{char} Data Type

The \texttt{char} data type is just large enough to store one character of your machine's character set. Variables of type \texttt{char} can be used to store numbers (-128 to 127 for a signed \texttt{char} and 0 to 255 for an unsigned \texttt{char}), but are generally used to hold the ASCII characters. Most C++ compilers use the ASCII character set, which has defined characters from 0 to 126. However, most compilers extend this range from 0 to 255 so as to accommodate the extended IBM character set. Appendix B contains a complete listing of the ASCII character set. For example, the decimal integers 0 to 9 are stored as decimal constants 48 to 57 and the lowercase letters a, b, ..., z are stored as decimal constants 97 to 122. Symbols such as +, - and # all have an equivalent ASCII character constant. At first it can be a little confusing to find out that characters are stored as integers which in turn are stored as binaries. Appendix B also lists the binary equivalents of the character set. Note that characters in the ASCII character set from 0 to 127, inclusive, could be represented as seven data bits instead of eight bits. However, since for most computers the byte (8 bits) is the primary unit, a variable of type \texttt{char} occupies one byte of memory.

For MS-DOS, characters 0-31 are reserved for control characters, 32-126 represent keys on the keyboard and 127-255 are the IBM extended characters. It is worth noting that the IBM extended character set is not part of the ASCII character set and may not be fully portable. MS-DOS also includes character sets or code pages. For example, the English and multilingual character sets are designated by the numbers 437 and 850 respectively. Note also that the Windows character sets differ from the MS-DOS character sets.

#### Character Constants

Character constants are written in C++ as a single character enclosed in apostrophes (or single quotes): 'C', 'p', '+'. Character constants are different from string constants enclosed within double quotes:

\begin{verbatim}
// ...
char char_const = 'C' ;          // character constant
// ...
cout << "this is a string constant" ;    // string constant
// ...
\end{verbatim}

When a character constant is encountered, the compiler converts it into its respective numeric code. The following example program illustrates character constants and variables:

\begin{verbatim}
// char.cpp
// illustrates the char data type
#include <iostream.h>   // C++ I/O
\end{verbatim}