5

Data Types and Operators

Key concepts

- Concept of Data Types
- C++ Data Types
- Fundamental Data Types
- Variables
- Operators
  - Arithmetic
  - Relational
  - Logical,
  - Input/Output
  - Assignment
- Expressions
  - Arithmetic
  - Relational
  - Logical
- Statements
  - Declaration
  - Assignment
  - Input /Output

In the previous chapter we learned the basic building blocks of C++ language. As we know, data processing is the main activity carried out in computers. All programming languages give importance to data handling. The input data is arranged and stored in computers using some structures. C++ has a predefined template for storing data. The stored data is further processed using proper operators. In this chapter, we will explore the main concepts of the C++ language like data types, operators, expressions and statements in detail.

5.1 Concept of data types

Consider the case of preparing the progress card of a student after an examination. We need data like admission number, roll number, name, address, scores in different subjects, the grades obtained in each subject, etc. Further, we need to display the percentage of marks scored by the student and the attendance in percentage. If we consider a case of scientific data processing, it may require data in the form of numbers representing the velocity of light (3×10^{8} m/s), acceleration due to gravity (9.8 m/s), electric charge of an electron (-1.6×10^{-19}), etc.

From the above cases, we can infer that data can be of different types like character, integer number, real number, string, etc. In the last chapter we saw that any valid character of C++ enclosed in single quotes represents character data in C++. Numbers without fractions represent integer data. Numbers with fractions represent floating point data and anything enclosed in double quotes represents a string data. Since the data to be dealt with are of many types, a programming language must provide ways and facilities to handle all types of data. C++ provides facilities to handle different types of data by providing data type names. **Data types** are the means to identify the nature of the data and the set of operations that can be performed on the data.

In Chapter 3, we used variables to refer data in algorithms. Variables are also used in programs for referencing data. When we write programs in the C++ language, variables are to be declared before their use. Data types are necessary to declare these variables.

5.2 C++ data types

C++ provides a rich set of data types. Based on nature, size and associated operations, they are classified as shown in Figure 5.1. Basically, they are classified into fundamental or built-in data types, derived data types and user-defined data types.

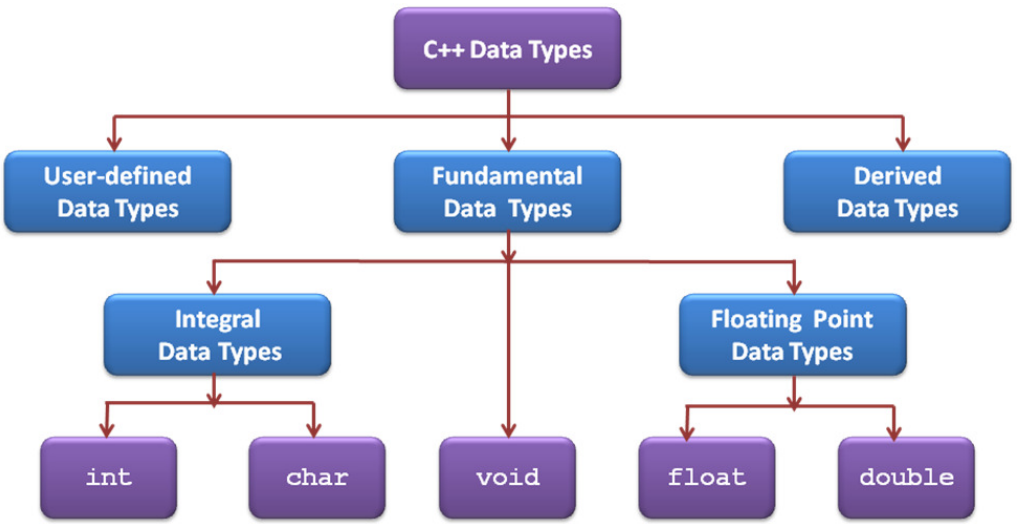

Fig. 5.1: Classification of C++ data types

Fundamental data types

Fundamental data types are defined in C++ compiler. They are also known as built-in data types. They are atomic in nature and cannot be further broken into small units. The

five fundamental data types in C++ are **char**, **int**, **float**, **double** and **void**. Among these, **int** and **char** come under integral data type as they can handle only integers. The numbers with fractions (real numbers) are generally known as floating type and are further divided into **float** and **double** based on precision and range.

User-defined data types

C++ is flexible enough to allow programmers to define their own data types. Structure (`struct`), enumeration (`enum`), `union`, `class`, etc. are examples for such data types.

Derived data types

Derived data types are constructed from fundamental data types through some grouping or alteration in the size. Arrays, pointers, functions, etc. are examples of derived data types.

5.3 Fundamental data types

Fundamental data types are basic in nature. They cannot be further broken into small units. Since these are defined in compiler, the size (memory space allocated) depends on the compiler. We use the compiler available in GCC and hence the size as well as the range of data supported by the data type are given accordingly. It may be different if you use other compilers like Turbo C++ IDE. The five fundamental data types are described below:

`int` data type (*for integer numbers*)

Integers are whole numbers without a fractional part. They can be positive, zero and negative. The keyword **int** represents integer numbers within a specific range. GCC allows 4 bytes of memory for integers belonging to `int` data type. So the range of values that can be represented by `int` data type is from -2147483648 to +2147483647. The data items 6900100, 0, -112, 17, -32768, +32767, etc. are examples of `int` data type. The numbers 2200000000 and -2147483649 do not belong to `int` data type as they are out of the allowed range.

`char` data type (*for character constants*)

Characters are the symbols covered by the character set of the C++ language. All letters, digits, special symbols, punctuations, etc. come under this category. When these characters are used as data they are considered as **char** type data in C++. We can say that the keyword `char` represents character literals of C++. Each `char` type data is allowed one byte of memory. The data items 'A', '+', '\t', '0', etc. belong to `char` data type. The `char` data type is internally treated as integers, because computer

recognises the character through its ASCII code. Character data is stored in the memory with the corresponding ASCII code. As ASCII codes are integers and need to be stored in one byte (8 bits), the range of `char` data type is from -128 to +127.

`float` data type (*for floating point numbers*)

Numbers with a fractional part are called floating point numbers. Internally, floating point numbers are stored in a manner similar to scientific notation. The number 47281.97 is expressed as 0.4728197×10^5 in scientific notation. The first part of the number, 0.4728197 is called the mantissa. The power 5 of 10 is called exponent. Computers typically use exponent form (*E notation*) to represent floating point values. In E notation, the number 47281.97 would be 0.4728197E5. The part of the number before the E is the mantissa, and the part after the E is the exponent. In C++, the keyword **`float`** is used to denote such numbers. GCC allows 4 bytes of memory for numbers belonging to `float` data type. The numbers of this data type has normally a precision of 7 digits.

`double` data type (*for double precision floating point numbers*)

In some cases, floating point numbers require more precision. Such numbers are represented by **`double`** data type. The range of numbers that can be handled by `float` type is extended by this data type, because it consumes double the size of `float` data type. In C++, it is assured that the range and precision of `double` will be at least as big as `float`. GCC reserves 8 bytes for storing a double precision value. The precision of `double` data type is generally 15 digits.

`void` data type (*for null or empty set of values*)

The data type **`void`** is a keyword and it indicates an empty set of data. Obviously it does not require any memory space. The use of this data type will be discussed in the next chapter. Table 5.1 shows the size and range of the fundamental data types of C++ based on GCC. The size of fundamental data types decreases in the order `double`, `float`, `int` and `char`.

Name	Description	Size	Range
`char`	Character	1 byte	-128 to 127
`int`	Integer	4 bytes	-2147483648 to +2147483647
`float`	Floating point number	4 bytes	$-3.4\times10^{+/-38}$ to $+3.4\times10^{+/-38}$ with approximately 7 significant digits
`double`	Double precision floating point number	8 bytes	$-1.7\times10^{+/-308}$ to $+1.7\times10^{+/-308}$ with approximately 15 significant digits
`void`	Null data	0 bytes	Empty set

Table 5.1: Data types and their characteristics

The values listed above are only sample values to give you a general idea of how the types differ. The values for any of these entries may be different on your system

5.4 Variables

Memory locations are to be identified to refer data. **Variables** are the names given to memory locations. These are identifiers of C++ by which memory locations are referenced to store or retrieve data. The size and nature of data stored in a variable depends on the data type used to declare it. There are three important aspects for a variable.

i. Variable name

It is a symbolic name (identifier) given to the memory location through which the content of the location is referred to.

ii. Memory address

The RAM of a computer consists of collection of cells each of which can store one byte of data. Every cell (or byte) in RAM will be assigned a unique address to refer it. All the variables are connected to one or more memory locations in RAM. The base address of a variable is the starting address of the allocated memory space. In the normal situation, the address is given implicitly by the compiler. The address is also called the L-value of a variable. In Figure 5.2 the base address of the variable **Num** is 1001.

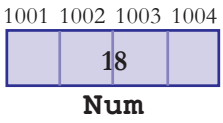

Fig 5.2 : Memory representation of a Variable

iii. Content

The value stored in the location is called content of the variable. This is also called the R-value of the variable. Type and size of the content depends on the data type of the variable.

Figure 5.2 shows the memory representaion of a variable. Here the variable name is **Num** and it consumes 4 bytes of memory at memory addresses 1001, 1002, 1003 and 1004. The content of this variable is 18. That is L-value of **Num** is 1001 and R-value is 18.

5.5 Operators

Operators are tokens constituted by predefined symbols that can trigger computer to carry out operations. The participants of an operation are called **operands**. An operand may be either a constant or a variable.

For example, a+b triggers an arithmetic operation in which + (addition) is the operator and a, b are operands. Operators in C++ are classified based on various criteria. Based on number of operands required for the operation, operators are classified into three. They are unary, binary and ternary.

Unary operators

A unary operator operates on a single operand. Commonly used unary operators are unary+ (positive) and unary− (negative). These are used to represent the signs of a number. If we apply unary+ operator on a signed number, the existing sign will not change. If we apply unary− operator on a signed number, the sign of the existing number will be negated. Examples of the use of unary operators are given in Table 5.2.

Variable x	Unary + +x	Unary- -x
8	8	-8
0	0	0
-9	-9	9

Table 5.2 : Unary operators

Some other examples of unary operators are increment (++) and decrement (−−) operators. We will discuss these operators later in this chapter.

Binary operators

Binary operators operate on two operands. Arithmetic operators, relational operators, logical operators, etc. are the commonly used binary operators.

Ternary operator

Ternary operator operates on three operands. The typical example is the conditional operator (? :).

The operations triggered by the operators mentioned above will be discussed in detail in the coming sections and some of them will be dealt with in Chapter 7. Based on the nature of the operation, operators are classified into arithmetic, relational, logical, input/output, assignment, short-hand, increment/decrement, etc.

5.5.1 Arithmetic operators

Arithmetic operators are defined to perform basic arithmetic operations such as addition, subtraction, multiplication and division. The symbols used for this are +, −, * and / respectively. C++ also provides a special operator % (modulus operator) for getting remainder during division. All these operators are binary operators. Note that + and − are used as unary operators too. The operands required for these operations are numeric data. The result of these operations will also be numeric. Table 5.3 shows some examples of binary arithmetic operations.

Variable x	Variable y	Addition x + y	Subtraction x - y	Multiplication x * y	Division x / y
10	5	15	5	50	2
-11	3	-8	-14	-33	-3.66667
11	-3	8	14	-33	-3.66667
-50	-10	-60	-40	500	5

Table 5.3 : Arithmetic operators

Modulus operator (%)

The modulus operator, also called as mod operator, gives the remainder value during arithmetic division. This operator can only be applied over integer operands. Table 5.4 shows some examples of modulus operation. Note that the sign of the result is the sign of first operand. Here in the table the first operand is **x**.

Variable x	Variable y	Modulus Operation x % y	Variable x	Variable y	Modulus Operation x % y
10	5	0	100	100	0
5	10	5	32	11	10
-5	11	-5	11	-5	1
5	-11	5	-11	5	-1
-11	-5	-1	-5	-11	-5

Table 5.4 : Operations using modulus operator

Check yourself

1. Arrange the fundamental data types in ascending order of size.
2. The name given to a storage location is known as _____.
3. Name a ternary operator in C++.
4. Predict the output of the following operations if x = -5 and y = 3 initially

   a. -x
   b. -y
   c. -x + -y
   d. -x - y
   e. x % -11
   f. x+y
   g. x%y
   h. x/y
   i. x * -y
   j. -x % -5

5.5.2 Relational operators

Relational operators are used for comparing numeric data. These are binary operators. The result of any relational operation will be either **True** or **False**. In C++, True is represented by **1** and False is represented by **0**. There are six relational operators in C++. They are **<** (less than), **>** (greater than), **==** (equal to), **<=** (less than or equal to), **>=** (greater than or equal to) and **!=** (not equal to). Note that equality checking requires two equal symbols (==). Some examples for the use of various relational operators and their results are shown in Table 5.5:

m	n	m<n	m>n	m<=n	m>=n	m!=n	m==n
12	5	0	1	0	1	1	0
-7	2	1	0	1	0	1	0
4	4	0	0	1	1	0	1

Table 5.5 : Operations using relational operators

5.5.3 Logical operators

Using relational operators, we can compare values. Examples are 3<5, num!=10, etc. These comparison operations are called relational expressions in C++. In some cases, two or more comparisons may need to be combined. In Mathematics we may use expressions like a>b>c. But in C++, it is not possible. We have to separate this into two, as a>b and b>c and these are to be combined using the logical operator **&&**, i.e. `(a>b)&&(b>c)`. The result of such logical combinations will also be either True or False (i.e. 1 or 0). The logical operators are **&&** (logical AND), **||** (logical OR) and **!** (logical NOT).

Logical AND (&&) operator

If two relational expressions El and E2 are combined using logical AND (**&&**) operation, the result will be 1 (True) only if both E1 and E2 have values 1 (True). In all other cases the result will be 0 (False). The results of evaluation of `&&` operation for different possible combination of inputs are shown in Table 5.6

E1	E2	E1&&E2
0	0	0
0	1	0
1	0	0
1	1	1

Table 5.6 : Logical AND

Examples: `10>5 && 15<25` evaluates to 1 (True)

`10>5 && 100<25` evaluates to 0 (False)

Logical OR (| |) operator

If two relational expressions El and E2 are combined using logical OR (| |) operations, the result will be 0 (False) only if both El and E2 are having value 0 (False). In all other cases the result will be 1 (True). The results of evaluation of || operation for different possible combination of inputs are shown in Table 5.7

E1	E2	E1 || E2
0	0	0
0	1	1
1	0	1
1	1	1

Table 5.7 : Logical OR

Examples: `10>5 || 100<25` evaluates to 1 (True)

`10>15 || 100<90` evaluates to 0 (False)

Logical NOT operator (!)

This operator is used to negate the result of a relational expression. This is a unary operation. The results of evaluation of ! operation for different possible inputs are as shown in Table 5.8.

E1	!E1
0	1
1	0

Table 5.8 : Logical NOT

Examples: `!(100<2)` evaluates to 1 (True)

`!(100>2)` evaluates to 0 (False)

5.5.4 Input / Output operators

Usually input operation requires user's intervention. In the process of input operation, the data given through the keyboard is stored in a memory location. C++ provides **>>** operator for this operation. This operator is known as ***get from*** or ***extraction*** operator. This symbol is constituted by two greater than symbols.

Similarly in output operation, data is transferred from RAM to an output device. Usually the monitor is the standard output device to get the results directly. The operator **<<** is used for output operation and is called ***put to*** or ***insertion*** operator. It is constituted by two less than symbols.

5.5.5 Assignment operator (=)

When we have to store a value in a memory location, assignment operator (**=**) is used. This is a binary operator and hence two operands are required. The first operand should be a variable where the value of the second operand is to be stored. Some examples are shown in Table 5.9.

Item	Description
a=b	The value of variable **b** is stored in **a**
a=3	The constant **3** is stored in variable **a**

Table 5.9 : Assignment operator

We discussed the relational operator == in Section 5.5.2. See the difference between these two operators. The = symbol assigns a value to a variable, whereas == symbol compares two values and gives True or False as the result.

5.6 Expressions

An expression is composed of operators and operands. The operands may be either constants or variables. All expressions can be evaluated to get a result. This result is known as the value returned by the expression. On the basis of the operators used, expressions are mainly classified into arithmetic expressions, relational expressions and logical expressions.

5.6.1 Arithmetic expressions

An expression in which only arithmetic operators are used is called arithmetic expression. The operands are numeric data and they may be variables or constants. The value returned by these expressions is also numeric. Arithmetic expressions are further classified into integer expressions, floating point (real) expressions and constant expressions.

Integer expressions

If an arithmetic expression contains only integer operands, it is called integer expression and it produces integer result after performing all the operations given in the expression. For example, if x and y are integer variables, some integer expressions and their results are shown in Table 5.10.

x	y	x + y	x / y	-x + x * y	5 + x / y	x % y
5	2	7	2	5	7	1
6	3	9	2	12	7	0

Table 5.10 : Integer expressions and their results

Note that all the above expressions produce integer values as the results.

Floating point expressions (Real expressions)

An arithmetic expression that is composed of only floating point data is called floating point or real expression and it returns a floating point result after performing all the

operations given in the expression. Table 5.11 shows some real expressions and their results, assuming that x and y are floating point variables.

x	y	x + y	x / y	-x + x * y	5 + x / y	x * x / y
5.0	2.0	7.0	2.5	5.0	7.5	12.5
6.0	3.0	9.0	2.0	12.0	7.0	12.0

Table 5.11 : Floating point expressions and their results

It can be seen that all the above expressions produce floating point values as the results.

In an arithmetic expression, if all the operands are constant values, then it is called ***Constant expression***. The expression `20+5/2.0` is an example. The constants like `15`, `3.14`, `'A'` are also known as constant expressions.

5.6.2 Relational expressions

When relational operators are used in an expression, it is called relational expression and it produces Boolean type results like True (1) or False (0). In these expressions, the operands are numeric data. Let us see some examples of relational expressions in Table 5.12.

x	y	x > y	x == y	x+y !=y	x-2 == y+1	x*y == 6*y
5.0	2.0	1(True)	0(False)	1(True)	1 (True)	0 (False)
6	13	0(False)	0(False)	1(True)	0 (False)	1 (True)

Table 5.12 : Relational expressions and their results

We know that arithmetic operators have a higher priority than relational operators. So when arithmetic expressions are used on either side of a relational operator, arithmetic operations will be carried out first and then the results are compared. The table contains some expressions in which both arithmetic and relational operators are involved. Though they contain mixed type of operators, they are called relational expressions since the final result will be either True or False.

5.6.3 Logical expressions

Logical expressions combine two or more relational expressions with logical operators and produce either True or False as the result. A logical expression may contain constants, variables, logical operators and relational operators. Let us see some examples in Table 5.13.

x	y	x>=y && x==20	x==5||y==0	x==y && y+2==0	!(x==y)
5.0	2.0	0 (False)	1 (True)	0 (False)	1 (True)
20	13	1 (True)	0 (False)	0 (False)	1 (True)

Table 5.13: Logical expressions and their results

Check yourself

1. Predict the output of the following operations if x=5 and y=3.
   a. x>=10 && y>=4
   b. x>=1 && y>=3
   c. x>5 && y>=2
   d. x<10 && y>2
   e. x<10 && y>x
   f. x>=10 || y>=4
   g. x>=1 || y>=3
   h. x>5 || y>=2
   i. x<10 || y>2
   j. x<10 || y>x
2. Predict the output if p=5, q=3, r=2
   a. p-q*r/2
   b. p*q/r
   c. p-q-r*2+p
   d. p+5*q+r*r/2

5.7 Statements

Can you recollect the learning hierarchy of a natural language? Alphabet, words, phrases, sentences, paragraphs and so on. In the learning process of C++ language we have covered character set, tokens and expressions. Now we have come to the stage where we start communication with the computer sensibly and meaningfully with the help of statements. **Statements** are the smallest executable unit of a programming language. C++ uses the symbol semicolon (;) as the delimiter of a statements. Different types of statements used in C++ are declaration statements, assignment statements, input statements, output statements, control statements, etc. Each statement has its own purpose in a C++ program. All these statements except declaration statements are executable statements as they possess some operations to be done by the computer. Executable statements are the instructions to the computer. The execution of control statements will be discussed in Chapter 7. Let us discuss the other statements.

5.7.1 Declaration statements

Every user-defined word should be defined in the program before it is used. We have seen that a variable is a user-defined word and it is an identifier of a memory location. It must be declared in the program before its use. When we declare a variable, we tell the compiler about the type of data that will be stored in it. The syntax of variable declaration is:

```
data_type <variable1>[, <variable2>, <variable3>, ...] ;
```

The `data_type` in the syntax should be any valid data type of C++. The syntax shows that when there are more than one variables in the declaration, they are separated by comma. The declaration statement ends with a semicolon. Typically, variables are declared either just before they are used or at the beginning of the program. In the syntax, everything given inside the symbols [and] are optional. The following statements are examples for variable declaration:

```
int rollnumber;
double wgpa,avg_score;
```

The first statement declares the variable `rollnumber` as **`int`** type so that it will be allocated four bytes of memory (as per GCC) and it can hold an integer number within the range from -2147483648 to +2147483647. The second statement defines the identifiers `wgpa` and `avg_score` as variables to hold data of **`double`** type. Each of them will be allocated 8 bytes of memory. The memory is allocated to the variables during the compilation of the program.

5.7.2 Assignment statements

When the assignment operator (**=**) is used to assign a value to a variable, it forms an assignment statement. It can take any of the following syntax:

```
variable = constant;
variable1 = variable2;
variable = expression;
```

The first statement assigns the constant to `variable`, the second assigns the value of `variable2` to the `variable1` and the last statement stores the result of the `expression` in `variable`. The following are some examples of assignment statements:

```
a = 15;          b = 5.8;
c = a + b;       d = (a + b)*(c + d);
```

The left hand side (LHS) of an assignment statement must be a variable. During execution, the expression at the right hand side (RHS) is evaluated first. The result is then assigned (stored) to the variable at LHS.

Assignment statement can be chained for doing multiple assignments at a time. For instance, the statement `x=y=z=13;` assigns the value 13 in three variables in the order of `z`, `y` and `x`. The variables should be declared before this assignment. If we assign a value to a variable, the previous value in it, if any, will be replaced by the new value.

5.7.3 Input statements

Input statement is a means that allows the user to store data in the memory during the execution of the program. We saw that the **get from** or **extraction** operator (**>>**) specifies the input operation. The operands required for this operator are the input device and a location in RAM where the data is to be stored. Keyboard being a standard console device, the stream (sequence) of data is extracted from the keyboard and stored in memory locations identified by variables. Since C++ is an object oriented language, keyboard is considered as the standard input stream device and is identified as an object by the name **`cin`** (pronounced as 'see in'). The simplest form of an input statement is:

```
streamobject >> variable;
```

Since we use keyboard as the input device, the `streamobject` in the syntax will be substituted by `cin`. The operand after the **>>** operator should strictly be a variable. For example, the following statement reads data from the keyboard and stores in the variable **`num`**.

```
cin >> num;
```

Figure 5.3 shows how data is extracted from keyboard and stored in the variable.

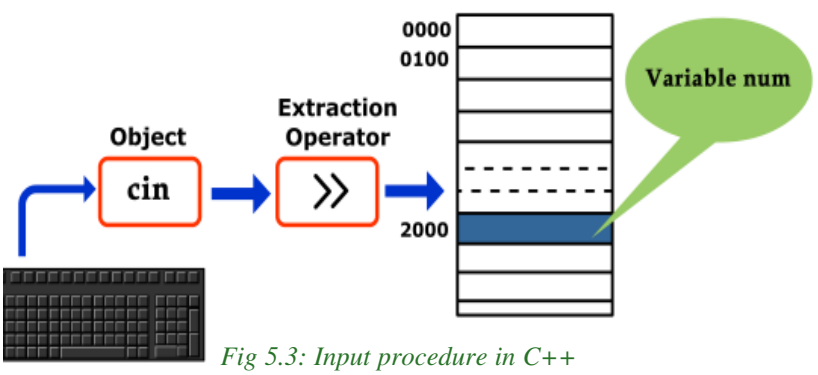

Fig 5.3: Input procedure in C++

5.7.4 Output statements

Output statements make the results available to the users through any output device. The **put to** or **insertion** operator (**<<**) is used to specify this operation. The operands in this case are the output device and the data for the output. The syntax of an output statement is:

```
streamobject << data;
```

The `streamobject` may be any output device and the data may be a constant, a variable or an expression. We use monitor as the commonly used output device and C++ identifies it as an object by the name `cout` (pronounced as 'see out'). The following are some examples of output statement with monitor as the output device:

```
cout << num;
cout << "hello friends";
cout << num+12;
```

The first statement displays the content in the variable **num**. The second statement displays the string constant `"hello friends"` and the last statement shows the value returned by the expression `num+12` (assuming that `num` contains numeric value). Figure 5.4 shows how data is inserted into the output stream object (monitor) from the memory location `num`.

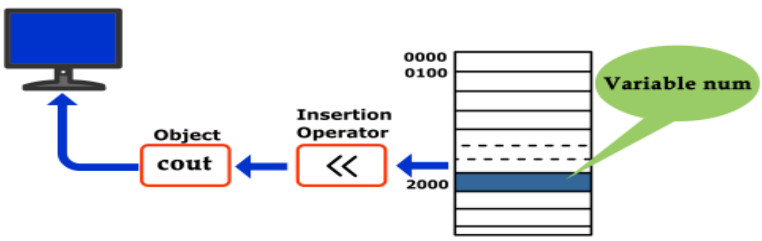

Fig. 5.4: Output procedure in C++

The tokens `cin` and `cout` are not keywords. They are predefined words that are not part of the core C++ language, and you are allowed to redefine them. They are defined in libraries required by the C++ language standard. Needless to say, using a predefined identifier for anything other than its standard meaning can be confusing and dangerous and such practices should be avoided. The safest and easiest practice is to treat all predefined identifiers as if they were keywords.

Cascading of I/O operators

Suppose you want to input three values to different variables, say x, y and z. You may use the following statements:

```
cin>>x;
cin>>y;
cin>>z;
```

But these three statements can be combined to form a single statement as given below:

```
cin>>x>>y>>z;
```

The multiple use of input or output operators in a single statement is called **cascading of I/O operators**. In the use of cascading of input operators, the values input are assigned to the variables from left to right. In the example `cin>>x>>y>>z;` the first value is assigned to x, the second to y and the third to z. While entering values to the variables x, y and z during execution the values should be separated by Space bar, Tab or Carriage return.

Similarly, if you want to display the contents of different variables (say x, y, z) use the following statement:

```
cout<<x<<y<<z;
```

If variables, constants and expressions appear together for output operations, the above technique can be applied as in the following example:

```
cout<<"The number is "<<z;
```

While cascading output operators, the values for the output will be retrieved from right to left. It is to be noted that both **<<** and **>>** operators cannot be used in a single statement.

Let us sum up

Data types are means to identify the type of data and associated operations handling it. Each data type has a specific size and a range of data. Data types are used to declare variables. Different types of operators are available in C++ for various operations. When operators are combined with operands (data), expressions are formed. There are mainly three types of expressions - arithmetic, relational and logical. Statements are the smallest executable unit of a program. Variable declaration statements define the variables in the program and they will be allocated memory space. The executable statements like assignment statements, input statements, output statements, etc. help giving instructions to the computer.

Learning outcomes

After the completion of this chapter the learner will be able to

- identify the various data types in C++.
- choose appropriate variables.
- experiment with various operators.
- apply the various I/O operators.
- write various expressions and statements.

Sample questions

Very short answer type

1. What are data types? List all predefined data types in C++?
2. What is the use of void data type?
3. What is a constant?
4. What is dynamic initialisation of variables?
5. Define operators.
6. What is meant by a unary operation?
7. What is declaration statement?
8. What is the input operator ">>" and output operator "<<" called ?
9. What will be the result of a = 5/3 if a is (i) float (ii) int ?
10. Write an expression that uses a relational operator to return true if the variable *total* is greater than or equal to *final*.
11. Given that i =4, j =5, k =2 what will be the result of following expressions?

    (i) j <k (ii) j <j (iii) j <= k (iv) i == j (v) i == k

    (vi) j > k (vii) j >= i (viii) j! = i (ix) j! = k (x) j <= k
12. What will be the order of evaluation for following expressions?

    (i) i+5>=j-6 (ii) s+10<p-2+2*q
15. What will be the result of the following if *ans* is 6 initially?

    (i) cout <<ans = 8 ; (ii) cout << ans == 8

Short answer type

1. What is a variable? List the two values associated with a variable.
2. Explain logical operators.
3. Find out the errors, if any, in the following C++ statements

   (i) cout<< "a=" a;
   (ii) m=5, n=l2;015
   (iii) cout << "x" ; <<x;
   (iv) cin >> y
   (v) cin >> "\n" >> y ;
   (vi) cout >> \n "abc"
   (vii) a = b + c
   (viii) break = x

4. What is the role of relational operators? Distinguish between == and =.

Long answer type

1. Explain the operators used in C++ in detail.
2. Explain different types of expressions in C++.