Data Types, Variables and Expressions

9.10 Introduction

Every programming language has two fundamental elements, viz., data types and variables. They are very essential elements to write even the most elementary programs. C++ provides a predefined set of data types for handling the data items. Such data types are known as fundamental or built-in data types. Apart from the built-in data types, a programmer can also create his own data types called as User-defined data types. In this chapter, we are going to learn about built-in data types.

9.11 Concept of Data types

Let us look at the following example,

Name = Ram

Age = 15

Average_Mark = 85.6

In the above example, Name, Age and Average_mark are the fields which hold the values such as Ram, 15 and 85.6 respectively.

In a programming language, **fields** are referred as **variables** and the **values** are referred to as **data**. Each data item in the above example looks different. That is, "Ram" is a sequence of alphabets and the other two data items are numbers. The first value is a whole number and the second one is a fractional number. In real-world scenarios, there are lots of different kinds of data we handle in our day-to-day life. The nature or type of the data item varies, for example distance (from your home to school), ticket fare, cost of a pen, marks, temperature, etc.,

In C++ programming, before handling any data, it should be clearly specified to the language compiler, regarding what kind of data it is, with some predefined set of data types.

9.12 C++ Data types

In C++, the data types are classified as three main categories

(1) Fundamental data types

(2) User-defined data types and

(3) Derived data types.

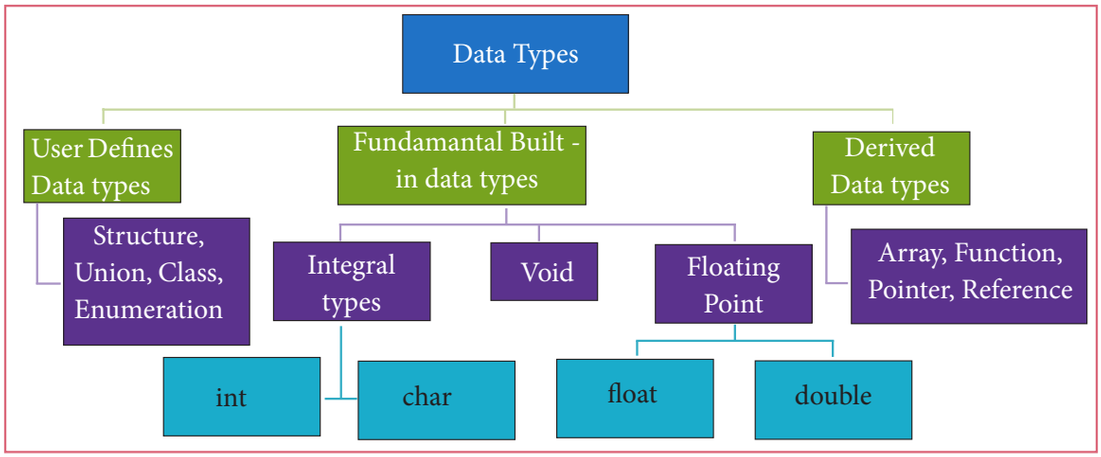


Figure 9.13 Data types in C++

In this chapter, we are going to learn about only the Fundamental data types.

In order to understand the working of data types, we need to know about variables. **T**he variables are the named memory locations to hold values of specific data types. In C++, the variables should be declared explicitly with their data types before they are actually used.

Syntax for declaring a variable:

<data type> <variable name>;

Example:

int num1;

To declare more than one variable which are of the same data type using a single statement, it can be declared by separating the variables using a comma.

Example:

int num1, num2, sum;

For example, to store your computer science marks first you should declare a variable to hold your marks with a suitable data type. Choosing an appropriate data type needs more knowledge and experience. Usually, marks are represented as whole numbers. Thus, the variable for storing the computer science marks should be of integer data type.

Example:

int comp_science_marks;

Now, one variable named **comp_science_marks** is ready to store your marks.

We will learn more about variables later in this chapter.

9.12.1 Introduction to fundamental Data types:

Fundamental (atomic) data types are predefined data types available with C++. There are five fundamental data types in C++: **char, int, float, double** and **void**. Actually, these are the keywords for defining the data types.

(1) int data type:

Integer data type accepts and returns only integer numbers. If a variable is declared as an **int**, C++ compiler allows storing only integer values into it. If you try to store a fractional value in an int type variable it will accept only the integer portion and the fractional part will be ignored.

For Example
int num=12;

num1 variable is declared as integer types. So, it can store integer value

(2) char data type:

Character data type accepts and returns all valid ASCII characters. Character data type is often said to be an integer type, since all the characters are represented in memory by their associated **ASCII Codes.** If a variable is declared as char, C++ allows storing either a character or an integer value.

Example 1:-

char ch='A';

cout<<ch ;

In the above code, **ch** is declared as a char type variable to hold a character. It displays the character A

Example 2:-

char ch='A';

ch=ch+1;

cout<<ch;

In the above statements, the value of **ch** is incremented by 1 and the new value is stored back in the same variable **ch**. (Remember that, arithmetic operations are carried out only on the numbers not with alphabets) so it displays B

Another program illustrates how **int** and **char** data types are working together.

Illustration 9.3: C++ program to get an ASCII value and display the corresponding character

```
#include <iostream>
using namespace std;
int main ()
{
        int n;
        char ch;
        cout << "\n Enter an ASCII code (0 to 255): ";
        cin >> n;
        ch = n;
        cout << "\n Equivalent Character: " << ch;
}
```

The output

```
Enter an ASCII code (0 to 255): 100
Equivalent Character: d
```

In the above program, variable **n** is declared as an int type and another variable **ch** as a char type. During execution, the program prompts the user to enter an ASCII value. If the user enters an ASCII value as an integer, it will be stored in the variable **n**. In the statement **ch = n**; the value of **n** is assigned into **ch**. Remember that, **ch** is a char type variable.

For example, if a user enters 100 as input; initially, 100 is stored in the variable **n**. In the next statement, the value of **n** i.e., 100 is assigned to **ch**. Since, **ch** is a char type; it shows the corresponding ASCII character as output. (Equivalent ASCII Character for 100 is d).

(3) float data type:

If a variable is declared as float, all values will be stored as floating point values.

There are two advantages of using float data types.

(1) They can represent values between the integers.

(2) They can represent a much greater range of values.

At the same time, floating point operations takes more time to execute compared to the integer type ie., floating point operations are slower than integer operations. This is a disadvantage of floating point operation.

For Example

```
float num=13.4;
```

In the above example, num variable is declared as float type .

(4) double data type:

This is for double precision floating point numbers. (precision means significant numbers after decimal point). The double is also used for handling floating point numbers. But, this type occupies double the space than float type. This means, more fractions can be accommodated in double than in float type. The double is larger and slower than type float. double is used in a similar way as that of float data type.

(5) void data type:

The literal meaning for void is 'empty space'. Here, in C++, the void data type specifies an empty set of values. It is used as a return type for functions that do not return any value.

Evaluate Yourself

1. What do you mean by fundemantal data types?
2. The data type char is used to represent characters. then why is it often termed as an integer type?
3. What is the advantage of floating point numbers over integers?
4. The data type double is another floating point type. Why is it treated as a distinct data type?
5. What is the use of void data type?

9.12.2 Memory representation of Fundamental Data types:

One of the most important reason for declaring a variable as a particular data type is to allocate appropriate space in memory. As per the stored program concept, every data should be accommodated in the main memory before they are processed. So, C++ compiler allocates specific memory space for each and every data handled according to

the compiler's standards.

The following Table 9.5 shows how much of memory space is allocated for each fundamental data type. Remember that, every data is stored inside the computer memory in the form of binary digits (See Unit I Chapter 2).

Table 9. 5 Memory allocation for Fundamental data types

Data type	Space in memory		Range of value
	in terms of bytes	in terms of bits	
char	1 byte	8 bits	-128 to 127
int	2 bytes	16 bits	-32,768 to 32,767
float	4 bytes	32 bits	3.4×10^{-38} to 3.4×10^{38} -1
double	8 bytes	64 bits	1.7×10^{-308} to 1.7×10^{308}-1

9.12.3 Data type modifiers:

Modifiers are used to modify the storing capacity of a fundamental data type except void type. Usually, every fundamental data type has a fixed range of values to store data items in memory. For example, int data type can store only two bytes of data. In reality, some integer data may have more length and may need more space in memory. In this situation, we should modify the memory space to accommodate large integer values. **Modifiers can be used to modify (expand or reduce) the memory allocation of any fundamental data type. They are also called as Qualifiers.**

There are four modifiers used in C++. They are:

(1) signed (2) unsigned (3) long (4) short

These four modifiers can be used with any fundamental data type. The following Table 9.6 shows the memory allocation for each data type with and without modifiers.

Integer type

Table 9.6 Memory allocation for Data types

Data type		Space in memory		Range of value
		in terms of bytes	in terms of bits	
short	short is a short name for **short int**	2 bytes	16 bits	-32,768 to 32,767
unsigned short	an integer number without minus sign.	2 bytes	16 bits	0 to 65535
signed short	An integer number with minus sign	2 bytes	16 bits	-32,768 to 32,767
Both short and signed short are similar				
int	An integer may or may not be signed	2 bytes	16 bits	-32,768 to 32,767

unsigned int	An integer without any sign (minus symbol)	2 bytes	16 bits	0 to 65535
signed int	An integer with sign	2 bytes	16 bits	-32,768 to 32,767
Both short and int are similar				
long	long is short name for long int	4 bytes	32 bits	-2147483648 to 2147483647
unsigned long	A double spaced integer without any sign	4 bytes	32 bits	0 to 4,294,967,295
signed long	A double spaced integer with sign	4 bytes	32 bits	-2147483648 to 2147483647

The above table clearly shows that an integer type accepts only 2 bytes of data whereas a long int accepts data that is double this size i.e., 4 bytes of data. So, we can store more digits in a long int. (long is a modifier and int is a fundamental data type)

char type

Table 9.7 Memory allocation for char Data types

Data type		Space in memory		Range of value
		in terms of bytes	in terms of bits	
char	Signed ASCII character	1 byte	8 bits	-128 to 127
unsigned char	ASCII character without sign	1 byte	8 bits	0 to 255
signed char	ASCII character with sign	1 byte	8 bits	-128 to 127

Floating point type

Table 9.8 Memory allocation for floating point Data types

Data type		Space in memory		Range of value
		in terms of bytes	in terms of bits	
float	signed fractional value	4 bytes	32 bits	3.4×10^{-38} to $3.4\times10^{38}-1$
double	signed more precision fractional value	8 bytes	64 bits	1.7×10^{-308} to $1.7\times10^{308}-1$
long double	signed more precision fractional value	10 bytes	80 bits	3.4×10^{-4932} to $1.1\times10^{4932}-1$

Memory allocation is subjected to vary based on the type of compiler that is being used. Here, the given values are as per the **Turbo C++** compiler. **Dev C++** provides some more space to **int** and **long** double types. Following Tables 9.9 shows the difference between Turbo C++ and Dev C++ allocation of memory.

Table 9.9 Memory allocation of Turbo C++ and Dev C++

Data type	Memory size in bytes	
	Turbo C++	Dev C++
short	2	2
unsigned short	2	2
signed short	2	2
int	2	**4**
unsigned int	2	**4**
signed int	2	**4**
long	4	4
unsigned long	4	4
signed long	4	4
char	1	1
unsigned char	1	1
signed char	1	1
float	4	4
double	8	8
long double	10	**12**

Since, Dev C++ provides 4 bytes to int and long, any one of these types can be used to handle bigger integer values while writing programs in Dev C++.

Note: sizeof( ) is an operator which gives the size of a data type.

Number Suffixes in C++

There are different suffixes for integer and floating point numbers. Suffix can be used to assign the same value as a different type. For example, if you want to store 45 in int, long, unsigned int and unsigned long int, you can use suffix letter **L** or **U** (either case) with 45 i.e. **45L** or **45U**. This type of declaration instructs the compiler to store the given values as long and unsigned. 'F' can be used for floating point values, example: 3.14F

9.13 Variables

Variables are user-defined names assigned to specific memory locations in which the values are stored. Variables are also identifiers; and hence, the rules for naming the identifiers should be followed while naming a variable. These are called as symbolic variables because these are named locations.

There are two values associated with a symbolic variable; they are **R**-value and **L**-value.

- R-value is data stored in a memory location
- L-value is the memory address in which the R-value is stored.

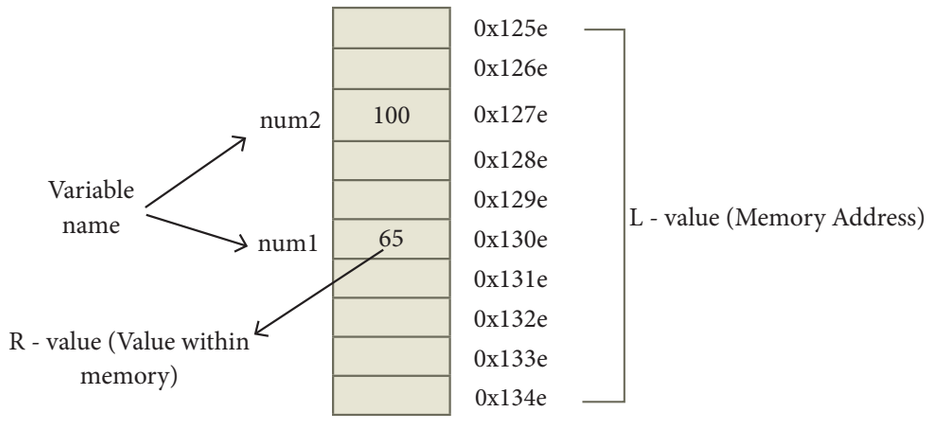


Figure 9.14 Memory allocation of a variable

Remember that, the memory addresses are in the form of Hexadecimal values

9.13.1 Declaration of Variables

Every variable should be declared before they are actually used in a program. Declaration is a process to instruct the compiler to allocate memory as per the type that is specified along with the variable name. For example, if you declare a variable as int type, in Dev C++, the compiler allocates 4 bytes of memory. Thus, every variable should be declared along with the type of value to be stored.

Declaration of more than one variable:

More than one variable of the same type can be declared as a single statement using a comma separating the individual variables.

Syntax:

<data type> <var1>, <var2>, <var3> …… <var_n>;

Example:

int num1, num2, sum;

In the above statement, there are three variables declared as int type. Which means, in **num1, num2** and **sum**, you can store only integer values.

For the above declaration, the C++ compiler allocates 4 bytes of memory (i.e. 4 memory boxes) for each variable.

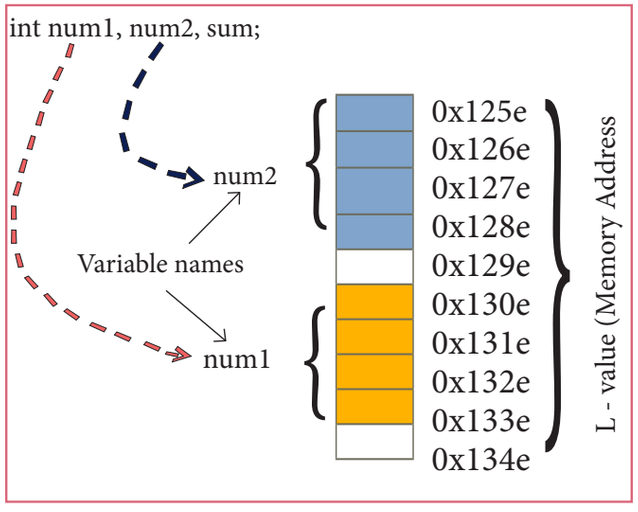


Figure 9.15 Memory allocation of int type variables

If you declare a variable without any initial value, the memory space allocated to that variable will be occupied with some unknown value. These unknown values are called as **"Junk"** or **"Garbage"** values.

```
#include <iostream>
using namespace std;
int main()
{
        int num1, num2, sum;
        cout << num1 << endl;
        cout << num2 << endl;
        cout << num1 + num2;
}
```

In the above program, some unknown values will be occupied in memory that is allocated for the variables **num1** and **num2** and the statement **cout << num1 + num2;** will display the sum of those unknown junk values.

9.13.2 Initialization of variables

Assigning an initial value to a variable during its declaration is called as "Initialization".

Examples:

int num = 100;

float pi = 3.14;

double price = 231.45;

Here, the variables num, pi, and price have been initialized during the declaration. These initial values can be later changed during the program execution.

Illustration 9.6 C++ Program to find the Curved Surface Area of a cylinder (CSA) (CSA = 2 pi r * h)

```
#include <iostream>
using namespace std;
int main()
{
   float pi = 3.14, radius, height, CSA;
   cout << "\n Curved Surface Area of a cylinder";
   cout << "\n Enter Radius (in cm): ";
   cin >> radius;
   cout << "\n Enter Height (in cm): ";
   cin >> height;
   CSA = (2*pi*radius)*height;
   system("cls");
   cout << "\n Radius: " << radius <<"cm";
   cout << "\n Height: " << height << "cm";
   cout << "\n Curved Surface Area of a Cylinder is " << CSA <<" sq. cm.";
}
```

Output:

```
Curved Surface Area of a cylinder
Enter Radius (in cm): 7
Enter Height (in cm): 20
Radius: 7cm
Height: 20cm
Curved Surface Area of a Cylinder is 879.2 sq. cm.
```

Variables that are of the same type can be initialized in a single statement.

Example:

int x1 = -1, x2 = 1, x3, n;

9.13.3 Dynamic Initialization

A variable can be initialized during the execution of a program. It is known as "**Dynamic initialization**". For example,

int num1, num2, sum;

sum = num1 + num2;

The above two statements can be combined into a single one as follows:

int sum = num1+num2;

This initializes sum using the known values of num1 and num2 during the execution.

Illustration 9.7 C++ Program to illustrate dynamic initialization

```
#include <iostream>
using namespace std;
int main()
{
        int num1, num2;
        cout << "\n Enter number 1: ";
        cin >> num1;
        cout << "\n Enter number 2: ";
        cin >> num2;
        int sum = num1 + num2; // Dynamic initialization
        cout << "\n Average: " << sum /2;
}
```

Output:

Enter number 1: 78
Enter number 2: 65
Average: 71

In the above program, after getting the values of num1 and num2, sum is declared and initialized with the addition of those two variables. After that, it is divided by 2.

Illustration 9.8: C++ program to find the perimeter and area of a semi circle

```
#include <iostream>
using namespace std;
int main()
{
        int radius;
        float pi = 3.14;
        cout << "\n Enter Radius (in cm): ";
        cin >> radius;
        float perimeter = (pi+2)*radius;   // dynamic initialization
        float area = (pi*radius*radius)/2; // dynamic initialization
        cout << "\n Perimeter of the semicircle is " << perimeter << " cm";
        cout << "\n Area of the semicircle is " << area << " sq.cm";
}
```

Output:

Enter Radius (in cm): 14
Perimeter of the semicircle is 71.96 cm
Area of the semicircle is 307.72 sq.cm

9.13.4 The Access modifier const

const is the keyword used to declare a constant. You already learnt about constant in the previous chapter. const keyword modifies / restricts the accessibility of a variable. So, it is known as Access modifier.

For example,

int num = 100;

The above statement declares a variable num with an initial value 100. However, the value of num can be changed during the execution. If you modify the above definition as **const int num = 100;** the variable num becomes a constant and its value will remain 100 throughout the program, and it can never be changed during the execution.

```
#include <iostream>
using namespace std;
int main()
{
        const int num=100;
        cout << "\n Value of num is = " << num;
        num = num + 1; // Trying to increment the constant
        cout << "\n Value of num after increment " << num;
}
```

In the above code, an error message will be displayed as **"Cannot modify the const object"** in Turbo compiler and **"assignment of read only memory num"** in Dev C++.

9.13.5 References

A reference provides an alias for a previously defined variable. Declaration of a reference consists of base type and an & (ampersand) symbol; reference variable name is assigned the value of a previously declared variable.

Syntax:

<type> <& reference_variable> = <original_variable>;

Illustration 9.9: C++ program to declare reference variable

```
#include <iostream>
using namespace std;
int main()
{
        int num;
        int &temp = num; //declaration of a reference variable temp
        num = 100;
cout << "\n The value of num = " << num;
cout << "\n The value of temp = " << temp;
}
```

The output of the above program will be

The value of num = 100

The value of temp = 100

Evaluate Yourself

1. What are modifiers? What is the use of modifiers?
2. What is wrong with the following C++ statement:
   long float x;
3. What is a variable ? Why is a variable called symblolic variable?
4. What do you mean by dynamic initialization of a variable? Give an exmple.
5. What is wrong with the following statement?
   const int x;

9.14 Formatting Output

Formatting output is very important in the development of output screens for easy reading and understanding. Manipulators are used to format the output of any C++ program. Manipulators are functions specifically designed to use with the insertion (<<) and extraction(>>) operators.

C++ offers several input and output manipulators for formatting. Commonly used manipulators are: **endl, setw, setfill, setprecision and setf**. In order to use these manipulators, you should include the appropriate header file. **endl** manipulator is a member of iostream header file. **setw, setfill, setprecision and setf** manipulators are members of iomanip header file.

endl (End the Line)

endl is used as a line feeder in C++. It can be used as an alternate to '\n'. In other words, endl inserts a new line and then makes the cursor to point to the beginning of the next line. There is a difference between endl and '\n', even though they are performing similar tasks.

- endl – Inserts a new line and flushes the buffer (Flush means – clean)
- '\n' - Inserts only a new line.

Example:

```
cout << "\n The value of num = " << num;
cout << "The value of num = " << num <<endl;
```

Both these statements display the same output.

setw ( )

setw manipulator sets the **width of the field** assigned for the output. The field width determines the minimum number of characters to be written in output.

Syntax:

setw(number of characters)

Illustration 9.10: Program to Calculate Net Salary

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
        float basic, da, hra, gpf, tax, gross, np;
        char name[30];
        cout << "\n Enter Basic Pay: ";
        cin >> basic;
        cout << "\n Enter D.A : ";
        cin >> da;
        cout << "\n Enter H.R.A: ";
        cin >> hra;
        gross = basic+da+hra; // sum of basic, da and hra
        gpf = (basic+da) * 0.10; // 10% 0f basic and da
        tax = gross * 0.10; //10% of gross pay
        np = gross - (gpf+tax); //netpay = earnings - deductions
        cout << setw(25) << "Basic Pay : " << setw(10)<< basic<< endl;
        cout << setw(25) << "Dearness Allowance : "<< setw(10)<<da<< endl;
        cout<<setw(25)<<"House Rent Allowance : "<<setw(10)<< hra<<endl;
        cout << setw(25) << "Gross Pay : " << setw(10) << gross << endl;
        cout << setw(25) << "G.P.F : " << setw(10) << gpf << endl;
        cout << setw(25) << "Income Tax : " << setw(10)<< tax << endl;
        cout << setw(25) << "Net Pay : " << setw(10) << np << endl;
}
```

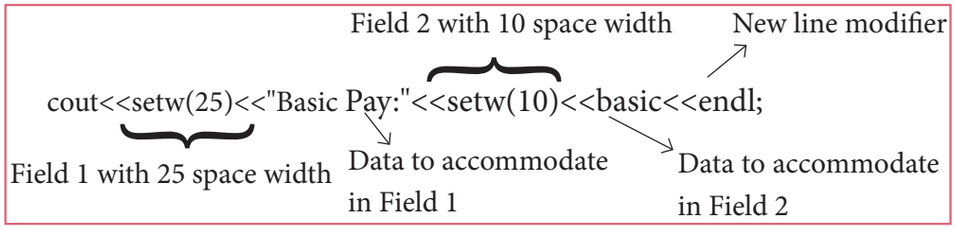
The output will be,

```
Enter Basic Pay: 12000
Enter D.A : 1250
Enter H.R.A : 1450
                    Basic Pay  :      12000
           Dearness Allowance  :       1250
       House Rent Allowance    :       1450
                    Gross Pay  :      14700
                        G.P.F  :       1325
                   Income Tax  :       1470
                       Net Pay :      11905
```

(HOT: Try to make multiple output statements as a single cout statement)

In the above program, every output statement has two setw( ) manipulators; first setw (25) creates a filed with 25 spaces and second setw(10) creates another field with 10 spaces. When you

represent a value to these fields, it will show the value within the field from right to left.



In field1 and field 2, the string "Basic Pay: " and the value of basic pay are shown as given in Figure 9.16 below.

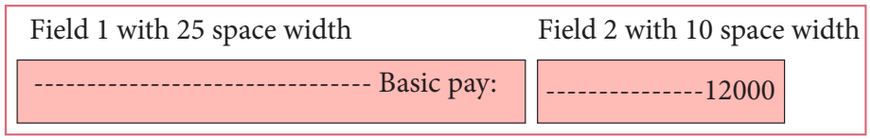


Figure 9.16 setw( ) function

setprecision ( )

This is used to display numbers with fractions in specific number of digits.

Syntax:

setprecision (number of digits);

Example:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{       float hra = 1200.123;
        cout << setprecision (5) << hra; }
```

In the above code, the given value 1200.123 will be displayed in 5 digits including fractions. So, the output will be **1200.1**

setprecision ( ) prints the values from left to right. For the above code, first, it will take 4 digits and then prints one digit from fractional portion.

setprecision can also be used to set the number of decimal places to be displayed. In order to do this task, you will have to set an ios flag within **setf()** manipulator. This may be used in two forms: (i) **fixed** and (ii) **scientific**

These two forms are used when the keywords fixed or scientific are appropriately used before the setprecision manipulator.

Example:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{       cout.setf(ios::fixed);
        cout << setprecision(2)<<0.1; }
```

In the above program, ios flag is set to **fixed** type; it prints the floating point number in fixed notation. So, the output will be, 0.10

cout.setf(ios::scientific);

cout << setprecision(2) << 0.1;

In the above statements, ios flag is set to **scientific type;** it will print the floating point number in scientific notation. So, the output will be, 1.00e-001

9.15 Expression

An expression is a combination of operators, constants and variables arranged as per the rules of C++. It may also include function calls which return values. (Functions will be learnt in upcoming chapters).

An expression may consist of one or more operands, and zero or more operators to produce a value. In C++, there are seven types of expressions, and they are:

(i) Constant Expression (ii) Integer Expression
(iii) Floating Expression (iv) Relational Expression
(v) Logical Expression (vi) Bitwise Expression
(vii) Pointer Expression

SN	Expression	Description	Example
1	Constant Expression	Constant expression consist only constant values	int num=100;
2	Integer Expression	The combination of integer and character values and/or variables with simple arithmetic operators to produce integer results.	sum=num1+num2; avg=sum/5;
3	Float Expression	The combination of floating point values and/or variables with simple arithmetic operators to produce floating point results.	Area=3.14*r*r;
4	Relational Expression	The combination of values and/or variables with relational operators to produce bool(true means 1 or false means 0) values as results.	x>y; a+b==c+d;
5	Logical Expression	The combination of values and/or variables with Logical operators to produce bool values as results.	(a>b)&& (c==10);
6	Bitwise Expression	The combination of values and/or variables with Bitwise operators.	x>>3; a<<2;
7	Pointer Expression	A Pointer is a variable that holds a memory address. Pointer variables are declared using (*) symbol.	int *ptr;

Table 9.10 : Types of Expressions

9.16 Type Conversion

The process of converting one fundamental data type into another is called as "Type Conversion". C++ provides two types of conversions.

(1) Implicit type conversion

(2) Explicit type conversion.

(1) Implicit type conversion:

An Implicit type conversion is a conversion performed by the compiler automatically. So, implicit conversion is also called as **"Automatic conversion".**

This type of conversion is applied usually whenever different data types are intermixed in an expression. If the type of the operands differ, the compiler converts one of them to match with the other, using the rule that the "smaller" type is converted to the "wider" type, which is called as **"Type Promotion".**

For example:

```
#include <iostream>
using namespace std;
int main()
{
        int a=6;
        float b=3.14;
        cout << a+b;
}
```

In the above program, operand **a** is an int type and **b** is a float type. During the execution of the program, int is converted into a float, because a float is wider than int. Hence, the output of the above program will be: **9.14**

The following Table 9.11 shows you the conversion pattern.

RHO / LHO	**char**	**short**	**int**	**long**	**float**	**double**	**long double**
char	int	int	int	long	float	double	long double
short	int	int	int	long	float	double	long double
int	int	int	int	long	float	double	long double
long	long	long	long	long	float	double	long double
float	float	float	float	float	float	double	long double
double	double	double	double	double	double	double	long double
long double	long double	long double	long double	long double	long double	long double	long double

(RHO – Right Hand Operand; LHO – Left Hand Operand)

Table 9.11: Implicit conversion of mixed operands

(2) Explicit type conversion

C++ allows explicit conversion of variables or expressions from one data type to another specific data type by the programmer. It is called as **"type casting".**

Syntax:

(type-name) expression;

Where type-name is a valid C++ data type to which the conversion is to be performed.

Example:

```
#include <iostream>
using namespace std;
int main( )
{
        float varf=78.685;
        cout << (int) varf;
}
```

In the above program, variable **varf is** declared as a **float** with an initial value 78.685. The value of **varf** is explicitly converted to an **int** type in cout statement. Thus, the final output will be 78.

During explicit conversion, if you assign a value to a type with a greater range, it does not cause any problem. But, assigning a value of a larger type to a smaller type may result in loosing or loss of precision values.

S.No	Explicit Conversion	Problem
1	double to float	Loss of precision. If the original value is out of range for the target type, the result becomes undefined
2	float to int	Loss of fractional part. If original value may be out of range for target type, the result becomes undefined
3	long to short	Loss of data

Table 9.12 – Explicit Conversion Problems

Example:

```
#include <iostream>
using namespace std;
int main()
{
        double varf=178.25255685;
        cout << (float) varf << endl;
        cout << (int) varf << endl;
}
```

Output:

178.253
178

Evaluate Yourself

1. What is meant by type conversion?
2. How implicit conversion is different from explicit conversion?
3. What is the difference between endl and \n?
4. What is the use of references?
5. What is the use of setprecision ( ) ?

Hands on practice:

1. Write C++ programs to interchange the values of two variables.
   a. Using the third variable
   b. Without using third variable
2. Write C++ programs to do the following:
   a. To find the perimeter and area of a quadrant.
   b. To find the area of triangle.
   c. To convert the temperature from Celsius to Fahrenheit.
3. Write a C++ to find the total and percentage of marks you secured from 10th Standard Public Exam. Display all the marks one-by-one along with total and percentage. Apply formatting functions.

Points to Remember

- Every programming language has two fundamental elements, viz., data types and variables.
- In C++, the data types are classified as three main categories (1) Built-in data types (2) User-defined data types (3) Derived data types.
- The variables are the named space to hold values of certain data type.
- There are five fundamental data types in C++: char, int, float, double and void.
- C++ compiler allocates specific memory space for each and every data handled according to the compiler's standards.
- Variables are user-defined names assigned to a memory location in which the values are stored.
- Declaration is a process to instruct the compiler to allocate memory as per the type specified along with the variable name.
- Manipulators are used to format output of any C++ program. Manipulators are functions specifically designed to use with the insertion (<<) and extraction(>>) operators.
- An expression is a combination of operators, constants and variables arranged as per the rules of C++.
- The process of converting one fundamental data type into another is called as "Type Conversion". C++ provides two types of conversions (1) Implicit type conversion and (2) Explicit type conversion.