Lecture 5:-

Variables

Data values in a program are usually stored in ***variables***, which are named locations in memory capable of holding a particular data type. You have already seen examples of variables in the **PowersOfTwo** program and are almost certainly familiar with the basic concept from your earlier programming experience. The purpose of this section is to outline the rules for using variables in C++.

Variable declarations

In C++, you must ***declare*** each variable before you use it. The primary function of declaring a variable is to make an association between the ***name*** of the variable and the ***type*** of value that variable contains. The placement of the declaration in a program also determines the ***scope*** of the variable, which is the region in which that variable is accessible.

The usual syntax for declaring a variable is

*type namelist***;**

where *type* indicates the data type and *namelist* is a list of variable names separated by commas. In most cases, each declaration introduces a single variable name. For example, the function **main** in **PowersOfTwo** begins with the line

int limit;

which declares the variable **limit** to be of type **int**. You can, however, declare several variable names at once, as in the following declaration, which declares three variables named **n1**, **n2**, and **n3**:

double n1, n2, n3;

In this case, the variables are each declared to be of type **double**, which is the type C++ uses to represent numbers that can have fractional parts. The name *double* is

short for *double-precision floating-point,* but there is no reason to worry about what all those terms mean.

Data types

Each variable in a C++ program contains a value constrained to be of a particular type. You set the type of the variable as part of the declaration. So far, you have seen variables of type **int** and **double**, but these types merely scratch the surface of the types available in C++. Programs today work with many different data types, some of which are built into the language and some of which are defined as part of a particular application. Learning how to manipulate data of various types is an essential part of mastering the basics of any language, including C++.

The concept of a data type

In C++ , every data value has an associated data type. From a formal perspective, a
data type is defined by two properties: a ***domain,*** which is the set of values that belong to that type, and a ***set of operations,*** which defines the behavior of that type. For example, the domain of the type **int** includes all integers
. . . -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5 . . .
and so on, up to the limits established by the hardware of the machine. The set of operations applicable to values of type **int** includes, for example, the standard arithmetic operations like addition and multiplication. Other types have a different domain and set of operations.

As you will learn in the later chapters in this course, much of the power of modern programming languages like C++ comes from the fact that you can define new data types from existing ones. To get that process started, C++ includes several fundamental types that are defined as part of the language. These types, which act as the building blocks for the type system as a whole, are called ***atomic*** or ***primitive types***. These predefined types are grouped into five categories—integer, floating-point, Boolean, character, and enumerated types—which are discussed in the sections that follow.

Expressions

Whenever you want a program to perform calculations, you need to write an expression that specifies the necessary operations in a

form similar to that used for expressions in athematics. For example, suppose that you wanted to solve the quadratic equation
a*x*2 + b*x* + c = 0
As you know from high-school mathematics, this equation has two solutions given by the formula
x =
The first solution is obtained by using + in place of the ± symbol; the second is obtained by using – instead. In C++, you could compute the first of these solutions by writing the following expression:

```
(-b + sqrt(b * b - 4 * a * c)) / (2 * a)
```

There are a few differences in form: multiplication is represented explicitly by a *****, division is represented by a ***/***, and the square root function (which comes from a library called **<cmath>** described in detail in Chapter 2) is written using the name **sqrt** rather than a mathematical symbol. Even so, the C++ form of the expression captures the intent of its mathematical counterpart in a way that is quite readable, particularly if you've written programs in any modern programming language.
In C++, an expression is composed of terms and operators. A ***term***, such as the
variables **a**, **b**, and **c** or the constants 2 and 4 in the preceding expression, represents
a single data value and must be either a constant, a variable, or a function call. An
operator is a character (or sometimes a short sequence of characters) that indicates a
-*b* + *b*2 - 4*ac*
2*a*

computational operation. A list of the operators available in C++ appears in
Table 1-4. The table includes familiar arithmetic operators like **+** and **-** along with
several others that pertain only to types introduced in later chapters.