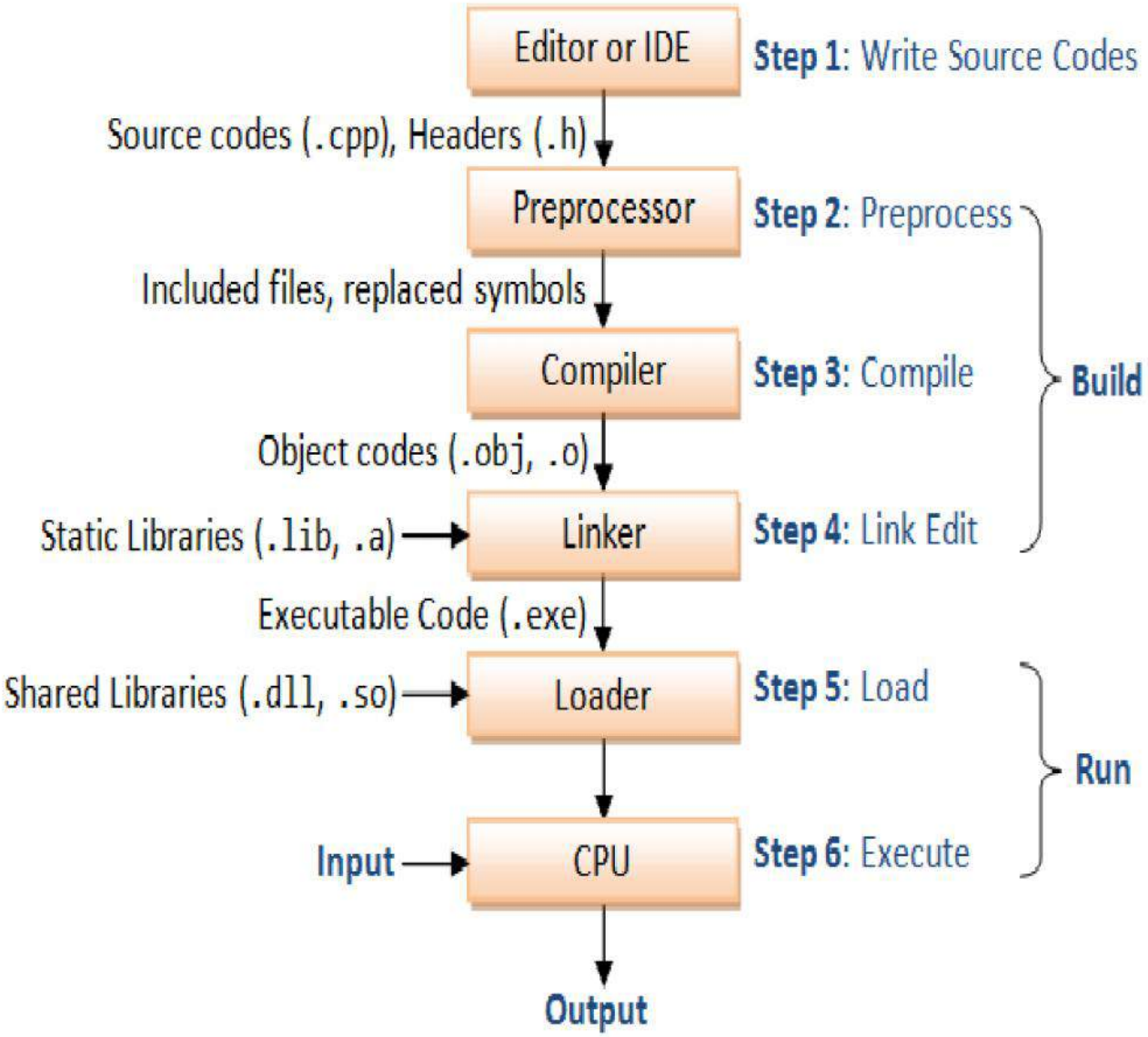
Chatpter 5

Review of C++

Objectives:

- **To understand the basics of C++**
- **To understand different control structures.**
- **To understand structured data types arrays,string, functions and structures.**



5.1 Review of C++ language

OPP OOP emphasizes on data. The ideology here is to unite both the data and the functions that operate on that data into a single unit called as **"object"**.

Therefore, an object is an identifiable entity with some characteristics and behavior.

OOP view any problem as object rather than as a procedure. For example, we can say 'mobile' is an object and its characteristics are color, weight, display, size etc. Its features include price, voice call, video call, memory size etc. Here **OOP considers the characteristics as data and features as functions**.

Another important concept with respect to OOP is the **'Class'**. A class serves as a plan or blueprint that specifies what data and what functions should be included in the objects of that class.

OOP characteristics

The characteristics of OOP are:

·Abstraction Data encapsulation Inheritance·
Polymorphism Polymorphism Dynamic binding ·
Message Passing

Modularity

Modularity is a concept where given problem is divided into sub-problems and the sub-problems are further divided. Each sub-problem is solved by writing a subprogram. Each subprogram is called a module.

Abstraction

Abstraction is an act which represents the essential features of an entity without including explanations or any background details about it.

OOP implements abstraction using classes as a list of abstract attributes.

Data Encapsulation

The binding of data and functions into a single unit called as the class is known as **encapsulation**.

The concept of insulating the data from direct access by the program is called **data hiding.**

Inheritance

Inheritance is the process by which objects of one class acquires the properties of the objects of another class.

Polymorphism

Polymorphism means taking more than one form. **Polymorphism** is the ability for a message to be processed in more than one form.

The process of making an operator to exhibit different behaviors in different instances is known as **operator overloading**.

Using a single function name to perform different types of tasks is known as **function overloading**.

Polymorphism allows objects to have different internal structures to share the same external interface. It supports to implement inheritance to a great extent.

Dynamic Binding

Binding means linking of a function call to the code to be executed when the function is called.

Dynamic Binding means that the code associated with a function call is not known until the time of the call at run-time. It is associated with polymorphism and inheritance.

Message Passing

The objects of a class can communicate can communicate with each other. The objects communicate with each other by sending and receiving messages. A message means a request for the execution of a function for an object. Therefore, a message invokes a function in the form of an object to generate the desired output.

For example, consider the message: stack1.push();

Here, we are invoking a function push of an object stack1. stack1 is an object of the class stack.

5.2 Fundamentals of C++

Bjarne Stroustrup developed C++ at AT & T Bell Laboratories in Murray Hill, New Jersey. He developed this language in early 1980's and was very much concerned in making C++ more efficient.

C++ character set

The following are the character set in C++.

Alphabets	A, B, …. , Za, b, …., z
Numerals	0, 1, 2, …., 9
Special characters:	+ - * % / \ . < > , = _ @ ! ^ & ~ { } [ ] ( ) etc

Tokens

A token is a smallest individual unit in a program or a lexical unit.

The most fundamental elements of a C++ program are "tokens". These elements help us to construct statements, definitions, declarations, and so on, which in turn helps us to construct complete programs. C++ has following tokens:

Identifiers	Keywords	Variables	Constants
Punctuators	Operators		

Identifiers

An identifier is a name given to the programming elements such as variables, arrays, functions etc. It can contain letter, digits and underscore. C++ is case sensitive and henceforth it treats uppercase and lowercase characters differently.

Exampe: Student _amount marks1 total_ score
STUDENT _AMOUNT RANK5 _Ad12

Keywords

Keyword is a predefined word that gives special meaning to the compiler. They are reserved words which can be used for their intended purpose and should not be used as normal identifier. Some of the keywords are given below:

and asm auto bool break case
catch char

Constants or Literals

Constant is an identifier whose value is fixed and does not change during the execution of the program.

Integer constants

Integer constants are numbers that has no fractional pars or exponent. It refers to the whole numbers. Integers always begin with a digit or + or -.

We can specify integer constants in decimal, octal, or hexadecimal form.

Unsigned constants:

To specify an unsigned type, use either the **u** or **U** suffix. To specify a long type, use either the **l** or **L** suffix.

Example 7.4: 328u 0x7FFFFFL 0776745ul;

Floating point constants

Floating-point constants are also called as **real constants.** These values contain decimal points (**.**) and can contain exponents. They are used to represent values that will have a fractional part and can be represented in two forms - fractional form and exponent form.

In the fractional form, the fractional number contains integer part and fractional part. A dot (.) is used to separate the integer part and fractional part.

Example: 65.05 125.5 -125.75

In the exponential form, the fractional number contains constants a mantissa and exponent. Mantissa contains the value of the number and the exponent contains the magnitude of the number. The **exponent**, if present, specifies the magnitude of the number as a **power of 10.**

Example: 7.6: 23.46e0 // $23.46 \times 10^0 = 23.46 \times 1 = 23.46$

23.46e1 // $23.46 \times 10^1 = 23.46 \times 10 = 234.6$

Character constants

Character constants are specified as single character enclosed in pair of single quotation marks. Single character constants are internally represented as ASCII codes.

For example: 'A' 'p' '5'

There is another class of characters which cannot be displayed on the screen (non-graphic characters) but they have special meaning and are used for special purpose. Such character constants are called as **escape sequences.**

For example: \' \" \\ \0 \a

String constants

A string zero or more characters enclosed by double quotation marks (**"**) is called a string or string constant.

String constants are treated as an array of char. By default, the compiler adds a special character called the **'null character'** (**'\0'**) at the end of the string to mark the end of the string.

For example: "Empress College" "Tumkur" "C++ Programming\0"

Punctuators

Punctuators are symbols in C++ and have syntactic and semantic meaning to the compiler. But do not by themselves specify an operation that yields a value. Some punctuators can be either alone or in combination. The following characters are considered as punctuators:

For example: ! % ^ & * ( ) –

C++ Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.

The operators can be either **'unary'** or **'binary'** or even **'ternary'** operators.

Unary operators

Unary operations have only one operand.

For example: ! & ~ * ++ — + -

Binary operators

The binary operators are those operators that operate on two operands. These operators can be arithmetic operators, relational, logical operators, bitwise, assignment operators.

Arithmetic Operators

The arithmetic operators are: + - * / %
These operators are used in arithmetic expressions

Relational Operators:

The relational operators are: == != > < >= <=

These operators are used in relational operations. The relational operations always give 0 (or False) or 1 (or True).

Logical Operators

The logical operators supported by C++ are: && || and !

Logical operators are used in logical expressions. Such expressions always give 0 (or False) or 1 (or True).

Bitwise Operators

Bitwise operator works on bits. The bitwise operators are: & | ^ ~ << >>

The Bitwise operators supported by C++ are listed in the following table 7.9.

Shorthand operators

We combine the assignment operators with arithmetic operators and bitwise operators. Such operators are called as shorthand operators

The shorthand operators are: += -= *= /= %=

&= |= ^=

Assignment Operator

It is an operator used to assign a value to a variable.

The syntax is variable = value or expression;

Example: n = 10; sum = n + 35;

Special Operators

Some special operators are there in C++. They are

sizeof() comma(,) dot(.) pointer(*)

Ternary operators

The ternary operators are those operators that operate on three or more operands. Ternary operator is also called as conditional operator.

? is the ternary operator. This operator can be used as an alternative to if-else statement.

Precedence of operators or Hierarchy of operators

If an expression contains multiple operators, the order in which operations are to be carried out is called hierarchy.

Example : x = 7 + 3 * 2;

Here x is assigned 13, not 20 because operator * has higher precedence than +. First multiply 3 and 2 and then adds into 7 and assign it to the variable x.

Type Conversion

Converting an expression of a given type into another type is known as type conversion.

Type conversions are of two types, they are

- Implicit conversion
- Explicit conversion

Implicit conversion

Implicit conversions do not require any type conversion operator. They are automatically performed when a value is copied to a compatible type. The C++ compiler will implicitly does conversion.

Explicit conversion will be performed by the user. The user can convert an expression from one type to another type. Such conversions are called as explicit conversion or type casting.

5.3 Structure of C++ Program

Include files
Class declarations
Member function declaration
main() function

Include files: This section is used to include all the preprocessor directives that are necessary for the program being written.

Class declaration: A class is a blueprint for the objects. It describes the data and functions that the object is going to use.

Member function declarations: This section defines or declares the user-defined functions that other functions or the main() function may use.

main() function: This is also a function that integrates all the other functions of the program. It contains the body of the function. The body should be enclosed within curled braces {and}.

The body contains two parts: Local declarations and executable statements. The local declaration refer to the declaration of all those variables that are used within the main() function. The executable statements are the statements that perform the required operations to solve the problem.

5.4 Libaray functions

C++ provides many built-in functions that save the programming time. They include mathematical functions, character functions, string functions, console input–output functions and some general standard library functions. Some of them are given below and also discussed in detail in Functions chapter.

Character Functions

All the character functions require **ctype.h** header file. The following table lists the function.

Some functions of character manipulation are given below:

isalpha() isdigit() isupper() islower() isspace() ispunct() tolower() toupper() toascii()

String Functions

The string functions are present in the **string.h** header file. Some string functions are given below:

Some of the functions are: strlen() strcat() strcpy() strrev() strupr() strlwr() strcmp() strcmpi()

Console I/O functions

The following are the list of functions is in **stdio.h** are:

getchar() putchar() gets() puts()

5.5 Data types

Variables

A variable is an identifier whose value is allowed to change during the execution of the program. A variable actually represent the name of the memory location.

Declaration of a variable

The syntax for declaring a variable is datatype variable_name;

Example: int n = 50;

Initializing a variable

The syntax to initialize a variable is: datatype variable_name = value;

Example : Let b be a variable declared of the type int. Then, int b = 100;

C++ compiler allows us to declare a variable at run time. **This is dynamic initialization.** They can be initialized anywhere in the program before they are used.

Example int a, b;

....

int temp = a;
a = b;
b = a;

5.5 Data types: C++ support the following data types.

Data types classification

There are two types of data types.

- Ø Simple or fundamental data types
- Ø Complex or Derived data types

The simple or fundamental data types are the primary data types which are not composed of any other data types.

The simple data types/fundamental data types include int, char, float double and void.

Modifiers

The data type modifiers are listed here: signed, unsigned, long and short.

Example: unsigned int b;

Derived data types

These data types are constructed using simple or fundamental data types. They include arrays, functions, pointers and references.

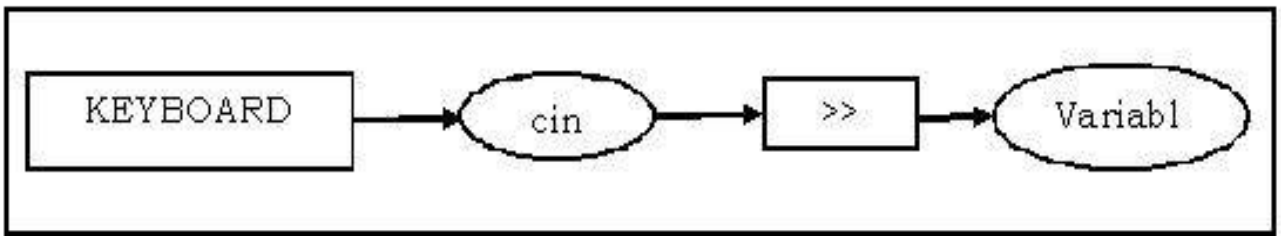
User defined data types

These data types are also constructed using simple or fundamental data types. Some user defined data types include structure, union, class and enumerated.

5.6 Input and output Operators

Input and output operators are used to perform input and output operations.

Input Operator ">>"

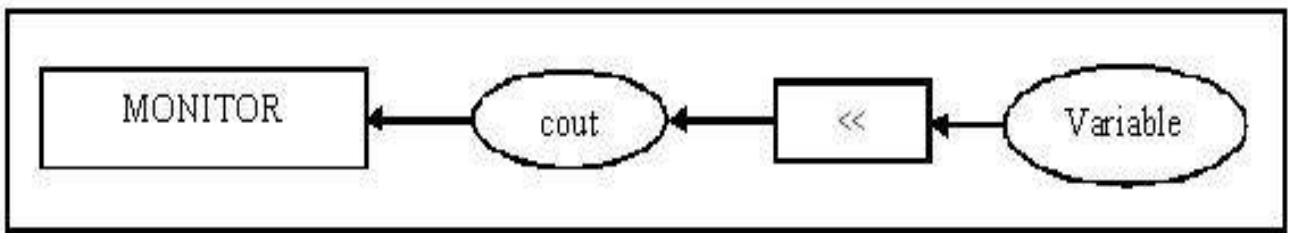


The standard input device is usually the keyboard. Input in C++ is done by using **stream extraction operator (>>)** on the cin stream. The operator must be followed by the variable that will store the data that is going to be extracted from the stream.

Example : int age;
cin>>age;

Output Operator "<<"

The standard output device is the screen (monitor) and outputting in C++ is done by using the object followed by the **stream insertion operator** which is written as << . cout stands for Console output.



Example: cout<< " Let us learn C++"; // prints Let us learn C++
//on the screen.

Cascading of I/O operators:

C++ supports the use of stream extraction (<<) and stream insertion (>>) operators many times in a single input (cin) and output (cout) statements. If a program requires more than one input variable then it is possible to input these variables in a single cin statement using multiple stream extraction operators. Similarly, when we want to output more than one result then this can be done using a single cout statement with multiple stream insertion operators. This is called as **cascading of input output operators**.

```
Example : cout<<" enter the value for x";
          cin>> x;
          cout<<" enter the value for y";
          cin>>y;
```

Instead of using cin statement twice, we can use a single cin statement and input the values for the two variables x and y using multiple stream extraction operator as shown below.

```
cout<<" enter the value for x and y";
cin>>x>>y;
```

Similarly, we can even output multiple results in a single cout statement using cascading of stream insertion operator as shown below.

```
cout<<" the sum of" <<x<<" and "<<y<<"="<<x+y;
```

5.7 Control Statements

C++ provides us **control structures, s**tatements that can alter the flow of a sequence of instructions. .

Compound statements

Compound statements or block is a group of statements which are separated by semicolons (;) and grouped together in a block enclosed in braces { and } is called a compound statement.

```
Example: {
           temp = a;
           a = b;
           b = temp;
         }
```

Types of control statements

C++ supports two basic control statements.

- Selection statements
- Iteration statements

Selection statements This statement allows us to select a statement or set of statements for execution based on some condition.

The different selection statements are:

i. if statement
ii. if-else statement
iii. nested statement
iv. switch statement

if statement

This is the simplest form of **if** statement. This statement is also called as **one-way branching.** This statement is used to decide whether a statement or a set of statements should be executed or not. The decision is based on a condition which can be evaluated to TRUE or FALSE.

Example if (n = = 100) cout<< " n is 100 ";

The if–else statement

This statement is also called as **two-way branching**. It is used when there are alternative statements need to be executed based on the condition. It executes some set of statements when the given condition is TRUE and if the condition is FALSE then other set of statements to be executed.

Nested-if statement

An if statement may contain another if statement within it. Such an if statement is called as nested if statement.

There are two forms of nested if statements:

Format I: if-else-if statement

This structure is also called as else-if ladder. This structure will be used to verify a range of values. This statement allows a choice to be made between different possible alternatives. A choice must be made between more than two possibilities.

Format II:

This structure contains an if-else statement within another if-else statement.

Switch statement

C++ has a built-in **multiple-branch selection** statement, switch. This successively tests the value of an expression against a list of integer or character constants. When a match is found, the statements associated with that code is executed.

Iteration statements or loops

Iteration statements are also called as loops. Loop is a statement that allows repeated execution of a set of instructions until certain condition is satisfied. This condition may be predefined or post-defined. Loops have a purpose to repeat a

statement or set of statements a certain number of times or some condition is fulfilled. We use three types of looping structures in C++.

- while loop
- do- while loop
- for loop

while loop

This looping structure is also called as **pre-tested looping structure.** This statement repeats the execution of a set of statements while the condition is TRUE.

Example:
```
c = 1;
while(c <= 10)
        cout<<setw(4)<<c;
```

do-while loop

This looping structure is also called as **post-tested looping structure**. Unlike while loop that test the loop condition at the beginning, the do-while loop checks the condition after the execution of the statements in it. This means that a do-while loop always executes at least once. Its functionality is exactly the same as the while loop, except that the condition in the do while loop is evaluated after the execution of statement, instead of before.

Example:
```
c = 1;
do
{
        cout<<setw(4)<<c;
} while(c <= 10);
```

The for loop

This statement is called as the **fixed execution looping statement**. It is normally used when we know in advance exactly how many times a set of statements should be repeatedly executed again and again. It provides initialization, loop-end-condition and increment/decrement process statements in a single line. When one is aware of fixed number of iterations, then this looping structure is best suited.

Jump statements or Transfer of control from within loop

Transfer of control within looping are used to

- Terminate the execution of loop
- Exiting a loop
- Half way through to skip the loop.

All these can be done by using **break, exit,** and **continue** statements.

5.8 ARRAYS

Array fundamentals

An array is collection of objects and all the objects have the same name. Each object is called an element. The elements are numbered as 0, 1, 2,..., n-1.

These numbers are called as indices or subscripts. These numbers are used to locate the positions of elements within the array.

The method of numbering the i^{th} element with index i-1 is called as **zero-based indexing**. That is, the element is always same as the number of "steps" from the initial element a[0] to that element. For example, the element a[3] is 3 steps from the element a[0].

Types of arrays

There are three types of arrays.

i. One-dimensional array
ii. Two-dimensional array
iii. Multi-dimensional array

One-dimensional array

It is an array in which each element is accessed by using **only one subscript**. The only one subscript represents the position of the element in the array.

Two-dimensional array

It is an array in which each element is accessed using 2-subsripts. The subscripts represent the position of element in the array.

Multi-dimensional array

A multidimensional array is an array of n-dimensions. In other words, an array of arrays is called a multidimensional array. A one-dimensional array of one-dimensional arrays is called a two-dimensional array; a one-dimensional array to two-dimensional arrays is called a three-dimensional array and so on.

One-dimensional array:

It is an array in which each element is accessed by using **only one subscript**. The only one subscript represents the position of the element in the array.

Declaration of one-dimensional array

Syntax datatype array-name[size];

Example : int marks[50];

Initialization of one-dimensional arrays

You can give values to each array element when the array is first defined.

Example : int a[5] = {9, -5, 6, 2, 8};

In the above example, value 9 is stored in a[0], value -5 is stored in a[1], value 6 is store in a[2], value 2 is stored in a[3] and value 8 is store in a[4].

Memory representation of one-dimensional arrays:

The elements of one-dimensional arrays are stored in contiguous memory locations.

Example : Consider the declaration, char a[5];

The element a[0] is allocated at a particular memory location, the element a[1] is allocated at next memory location and so forth. Since the array is of the type char, each element requires 1-byte.

Two-dimensional arrays

It is an array in which each element is accessed using 2-subsripts. The subscripts represent the position of the elements in the array.

The elements of two dimensional arrays are represented as rows and columns. To identify any element, we should know its row-number and column-number.

Declaration of two-dimensional array:

Syntax datatype array-name[row-size][column-size];

Example : int a[2][3];

Initialization of two-dimensional arrays

Example : int a[2][3] = {1, 2, 3, 4, 5, 6};

a is a two dimensional array which contains 2 rows and 3 columns and these assignments would be

a[0][0] = 1 a[0][1] = 2 a[0][2] = 3

a[1][0] = 4 a[1][1] = 5 a[1][2] = 6

If the values are missing in an initialize, they are automatically set to 0.

```
Example :   int  b[2][3] = {
                              {1, 2},
                              {3}
                           };
```

will initialize the first two elements to the first row, the next element to the second row. The remaining elements are automatically set 0.

b[0][0] = 1 b[0][1] = 2 b[0][2] = 0

b[1][0] = 3 b[1][1] = 0 b[1][2] = 0

Multi-dimensional array

A multidimensional array is an array of n-dimensions. In other words, an array of arrays is called a multidimensional array. A one-dimensional array of one-dimensional arrays is called a two-dimensional array; a one-dimensional array to two-dimensional arrays is called a three-dimensional array and so on.

5.9 FUNCTIONS

If the programs are complex and lengthy, they can be modularized into subprograms. The subprograms are called as functions. The subprograms can be developed independently, compiled and tested. They can be reused in other programs also.

A function is a named group of statements developed to solve a sub-problem and returns always a value to other functions when it is called.

Types of functions

There are two types of functions:

i. Library functions
ii. User-defined functions.

i. Library functions

A standard library is a collection of pre-defined functions and other programming elements which are accessed through header files.

Header files are the files containing standard functions that our programs may use. This chapter contains the information about some header files of C++ standard library and some functions of it. The header files should be written within angled brackets and its functions are included into our programs by #include directive.

User-defined functions

We can create our own functions or sub-programs to solve our problem. Such functions are normally referred to as **user defined functions.**

A user-defined function is a complete and independent program, which can be used (or invoked) by the main program or by the other sub-programs. The user-defined functions are written to perform definite calculations, after performing their task they send back the result to the calling program or sub-program.

Different header files

As said earlier, header files are the files containing standard functions that our programs may use. C++ contains many header files and is listed below.

stdio.h

This header file contains functions and macros to perform standard I/O operations. When we include the header file iostream.h, the header file stdio.h is automatically included into our program.

string.h

This header file declares functions to manipulate strings.

stdlib.h

This header file is used to declare conversion routines, search/sort routines and other miscellaneous things.

iostream.h

This header file contains C++ streams and i/o routines.

iomanip.h

This header file contains functions and macros for I/O manipulators for creating parameterized manipulations.

math.h

This header file declares prototypes for the mathematical functions and error handlers. The functions are used to perform mathematical calculations.

Mathematical library functions

C++ provides many mathematical functions. These functions can be used in mathematical expressions and statements. The various functions are ceil(), exp(), fabs(), floor(), log(), pow() etc.

Character and string functions

A character is any single character enclosed within single quotes. Some functions accept a character as argument. The argument is processed as an int by using its ASCII code. To use these functions, the header file **ctype.h** should be included.

Inputting single character

We can input a character using the function get( ).

```
char      ch;                  char  ch;
cin = get(ch);   OR   ch = cin.get( );
```

Outputting single character

```
cout.put(ch);
```

put() function is used to display single character.

The general form is

Example : `char ch; ch = cin.get(); cout.put(ch);`

String functions

A string is sequence of characters enclosed within double quotes. Strings are manipulated as one-dimensional array of characters and terminated by null ('\0') character. C++ provides many functions to manipulate strings. To use these functions, the header file **string.h** should be included.

```
char      string-name[size];
```

Declaring a string variable

The general form to declare a string is:

Ø string-name is the name of the string variable

- Size is the number of characters in the string. The size helps the compiler to allocate required number of memory locations to store the string.

Example : char st[50];

Initializing a string

Like other variables, strings can also be initialized when they are declared.

Example : char s[10] = "Karnataka";

There are only 9 characters in the string. The null character ('\0') is automatically appended to the end of the string.

Inputting a string

C++ provides the function getline() to read a string.

cin.getline(string, size);

Example cin.getline(st, 25);

Outputting a string

C++ provides the function write() to output a string.

cout.write(string, size);

Example : cout.write(st, 25);

Some string manipulation functions are given below:

strlen() function

This function returns the length of the string. i.e., the number of characters present in the string, excluding the null character.

The general form is variable = strlen(string);

Example 12.9: l = strlen("Empress"); Returns 7.

strcat() function

This function is used to concatenate two strings. The process of combining two strings to form a string is called as concatenation.

strcpy() function

A string cannot be copied to another string by using assignment statement. The function strcpy() is used to copy a string into another string.

strcmp() function

This function is used to alphabetically compare a string with another string. This function is **case-sensitive**. i.e., it treats the uppercase letters and lowercase letters as different.

strcmpi() function

This function is used to alphabetically compare a string with another string. This function is **not case-sensitive**. i.e., it treats the uppercase letters and lowercase letters as same.

strrev() function

This function is used to reverse the characters in a string.

5.10 USER DEFINED FUNCTIONS

Definition

User-defined function is a function defined by the user to solve his/her problem. Such a function can be called (or invoked) from anywhere and any number of times in the program.

Function definition or structure of user-defined function

Function-header

Any user-defined function has the following structure.

```
return-type-specifier  function-name(argument-list with declaration)
{
    Local-variable-declarations;
    Executable-statement-1;
    Body of the function
    Executable-statement-2;
            ..........
    Executable-statement-n;
    return(expression);
}
```

Ø **Return-type-specifier** is the data type of the value return by the function to anther function when it is called. The return-type-specifier can be char, int, float or void. The data type void is used when the function return no value to the calling function.

Ø **Function-name** is the name of the function. It is an identifier to identify the particular function in a program.

Ø **Argument-list with declaration** is the list of arguments or parameters or variables with their declaration. Each argument should be declared separately. Each declaration should be separated by comma. The list should be enclosed within parenthesis.

Ø The complete line is called the function header. Note that there is no semicolon at the end.

Ø **Local-variable declaration** is the declaration of the variables that are used within the function. Since these variables are local to the function, these variables are called as local variables.

Ø **Executable-statements** are the statements that perform the necessary operations to solve the problem. If the function is returning a value, a return

statement should be included. Otherwise, return statement is not necessary.

Ø Local declaration and executable statements are together called as **body of the function**. The body of the function should be enclosed within the curled braces.

Calling a function

variable = function-name(argument-list);

OR

variable = function-name();

A function can be called by specifying its name, followed by list of arguments enclosed within the parenthesis. The arguments should be separated by commas. If the function does not pass any arguments, then an empty pair of parenthesis should follow the name of the function.

The function call should be a simple expression such as an assignment statement or it may be part of a complex expression.

Example : big = biggest(a, b, c);

main() function

In C++, the main() function returns a value of type **int** to the operating system. If the program runs successfully, 0 is returned. Otherwise, a non-zero value is returned to the operating system, indicating that the program contains errors. If the main() function is not returning a value, the datatype **void** can be used as return-type-specifier.

The general form of main() function is:

```
int    main()                          void    main()
{                                      {
        Executable-statements;    OR           Executable-statements;
        return   0;                    }
}
```

Returning a value

When a function is called, the statements in the called function are executed. After executing the statements, the function returns a value to the calling function. The return statement is used to return a value. A function can return only one value or none to the calling function, for every function call.

The general form of return statement is:

return(expression); OR return 0;

Function prototypes

Like all variables are declared before they are used in the statements, the function should also be declared. The function prototype is a declaration of the

function in the calling function.

The general form of function prototype is

return-type-specifier function-name(type arg1, type arg2, ……);

OR

return-type-specifier function-name(type , type , ……);

Example : float volume(int x, float y, float z);
Or float volume(int, float, float);

Types of arguments

A variable in a function header is called an argument. The arguments are used to pass information from the calling function to the called function.

Actual arguments

The function call statement contains name of the function and list of arguments to be passed. These arguments or parameters are called as actual arguments. The arguments can be constants, variables or expressions. Actual arguments have values stored in them before the function call hence the name actual.

Example : In the function call g = gcd(a, b);

Formal arguments

The function header contains return-type-specifier, function name and list of arguments with their declaration. These arguments are called as formal arguments or dummy arguments. Formal arguments get their values from the actual arguments.

Example: In the function header int gcd(int x, int y)
x and y are the formal arguments.

Local variables

The variables declared inside function or block is said belong to that block. These variables are called as Local variables. Values of local variables are accessible only in that block. The function's formal arguments are also considered as local variables.

Global variables

The variables declared outside the function are called as global variables. These variables are referred by the same data type and same name throughout the program in both the calling function and called function. Whenever if some variables are to be treated as same value in both main() and in other functions, it is advisable to use global variables.

The availability of the values of global variables starts from the point of definition to the rest of the program.

Types of Functions :

There are 5 types of functions.

i. Functions with no arguments and no return values
ii. Functions with arguments and with no return values
iii. Functions with no arguments and with return values
iv. Functions with arguments and with return values
v. Recursive functions

Functions with no arguments and with no return values

In this method, the function simply performs an independent task. The function does not receive or send any arguments.

Example :

```
void  natural()
{
        for(int i=1; i <= 10; i++)
                cout<<setw(4)<<i;
}
```

Functions with arguments and with no return values

In this method, the function receives some arguments and does not return any value.

Example :

```
void  average(int x, int y, int z)
        {
                float sum, avg;
                sum = a + b + c;
                avg = sum/3.0;
                cout<<"Average = "avg<<endl;
        }
```

Functions with no arguments and with return values

In this method, the function receives no arguments but return a value.

Example :

```
int   greatest()
{
        if(a>b)
                return(a);
        return(b);
}
```

Functions with arguments and with return values

In this method, the function receives some arguments and returns a value.

Example :

```
float  interest(float p, float t, float r)
        {
                si = (p * t * r)/100;
                return(si);
        }
```

Recursive functions

Recursive function is a function that calls itself. The process of calling a function by itself is called as recursion.

Recursive functions must have one or more terminating conditions to terminate recursion. Otherwise, recursion will become infinite.

Passing default arguments to functions

In C++, to call a function we need not pass all the values for the arguments to a function from the calling function. It allows us to assign default values to the formal arguments.

Example : Consider the prototype

float interest (float amt, int time, **float rate = 0.15**);

0.15 is the default value provided to the argument rate.

The function call statement can be as follows:

si = interest(5000,5); // third argument is missing

Default values should be assigned only in the function prototype. It should not be repeated in the function definition.

Passing constant arguments

In C++, we can declare some arguments as constants. The compiler cannot modify the arguments marked as constants.

Example :
```
int strlen(const char *p);
int total( constint x, const int y);
```

Pass by value or Call by value

A function can be called by passing arguments from the calling function into the called function. Thus the data is transferred through argument list.

Pass by reference or call by reference

We can pass parameters to the function by using reference variables. When we pass arguments by reference, the formal arguments in the called function become the **aliases** to the actual arguments of the calling function. i.e., the called function is actually uses the original data with a different name.

Passing arrays to functions

To pass an array to a function, we just pass the name of the array to the function. i.e., we are referring the address of the first element of the array. Using this address, the function can access all the elements of the array.

Passing structures to functions

We can pass structures to functions as we pass other arguments. Structures are passed to functions in pass-by-value method. i.e., the function works with copy of the structures. The function can also return a structure after processing it.

Whenever we pass the address of the structure to the function, we should include the address-of (&) operator.

5.11 Structures

A structure is a collection of simple variables. The variables in a structure can be of same or different types: Some can be int, some can be float and so on. The data items in a structure are called the **members** of the structure.

Defining a structure

The process of defining a structure is equivalent to defining your own data type.

```
struct    structure-name
{
          datatype  member-name-1;
          datatype  member-name-2;
              ..............
          datatype  member-name-n;
};
```

Example: A structure definition to hold employee information.

```
struct employee
{
        int     idno;
        char    name[15];
        char    designation[10];
        float   salary;
};
```