

PAPER – II

UNIT – 1: ELEMENTS OF PROGRAMMING AND FUNCTIONS

Syllabus

Introduction: Basic Elements of Programming, Console I/O Operations,

Function: Function Prototyping, Call and Return By Reference, Inline Function, Default and Const Arguments, Function Overloading, Arrays, Manipulators and Enumeration.

1.1 INTRODUCTION ABOUT C++

C++ is an object oriented programming language. It was developed by Bjarne Stroustrup in 1983 at AT & T Bell laboratories in USA. It is an incremented version of C language. It can also be an extension of C language and superset of C language.

The facilities are available in C++ like Classes, Function overloading, Operator overloading, Encapsulation, Inheritance, Polymorphism, Abstraction etc.

In this language, object oriented libraries can be in-built.

1.2 ADVANTAGES OF C++

- 1) C++ is a superset of C language.
- 2) It is an extension of C language.
- 3) It is an incremented version of C language.
- 4) It is object oriented programming language
- 5) All programs can run in C++ compiler.
- 6) It includes facilities like classes, function, overloading, operator overloading, encapsulation, inheritance, Abstraction etc.
- 7) It supports polymorphism.
- 8) Object oriented libraries can be in-built by C++.
- 9) C++ program can be easily implemented, maintained and expanded.

1.3 CHARACTER SET

A character is an alphabet, digit or special symbols. It is used to represent information. The collection of all such characters is called character set. Every language has its own character set. C++ use valid character set is as given

Alphabet (Upper alphabets) A, B, C,.....,Z or
(Lower alphabets) a, b, c, -----, z

Digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Special symbols ~ ' ! @ # % ^ & * () _ -- + | \ { } [] : ; " < > , . ? /

Formatting characters backspace, horizontal tab, vertical tab, form feed and carriage return

1.4 TOKENS

Token is a collection of elements which are identified by compiler. Also token is the smallest individual unit in a program. C++ uses following types of tokens -

- 1) Keywords
- 2) Identifiers
- 3) Literals
- 4) Data types
- 5) Constants
- 6) Variables
- 7) Operators

1. Keywords

Keywords are also called reserved words. There are total 48 keywords are available. These keywords cannot be used as variable names.

The list of C++ keywords are as –

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	register	typedef
class	friend	return	union
const	goto	short	unsigned
continue	if	signed	virtual
default	inline	sizeof	void
delete	int	static	volatile
do	long	struct	while

2. Identifiers

Identifier is also known as symbolic name. It refers to the names of variables, functions, arrays, and classes etc.

Rules for creating identifiers –

- a) An identifier can consist of alphabets, digits and/or underscores.
- b) It must not start with digit.
- c) C++ is case sensitive that is Upper case and Lower case letter are distinct.
- d) It should not be a reserved / keyword words.

3. Literals

Literals are also known as Constants. Literals are data items that never change their value during the execution of the program. The types of literals are as

- a) Integer-Constants
- b) Character-constants
- c) Floating-constants
- d) Strings-constants

a) Integer Constants

Integer constants are whole number without any fractional part.

There three types of integer constants-

1) Decimal integer constants

It consists of sequence of digits and should not begin with 0 (zero).

For example 124, - 179, +108.

2) Octal integer constants

It consists of sequence of digits starting with 0 (zero).

For example. 014, 012

3) Hexadecimal integer constant

It consists of sequence of digits preceded by ox or OX.

b. Character constants

A character constant must contain one or more characters and must be enclosed in single quotation marks.

For example 'A', '9', etc.

An escape sequence represents a single character.

Escape sequences characters

Escape sequence is a character constant of special characters.

Escape sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

c. Floating constants

Floating constant is also called real constants. They are numbers having fractional parts. They may be written in fractional form or

exponent form. A real constant in fractional form consists of signed or unsigned digits including a decimal point between digits.

For example 3.0, -17.0, -0.627 etc.

d. String constants

A sequence of character enclosed within double quotes is called string constants. String constant is by default (automatically) added with a special character '\0' which denotes the end of the string. Therefore the size of the string is increased by one character.

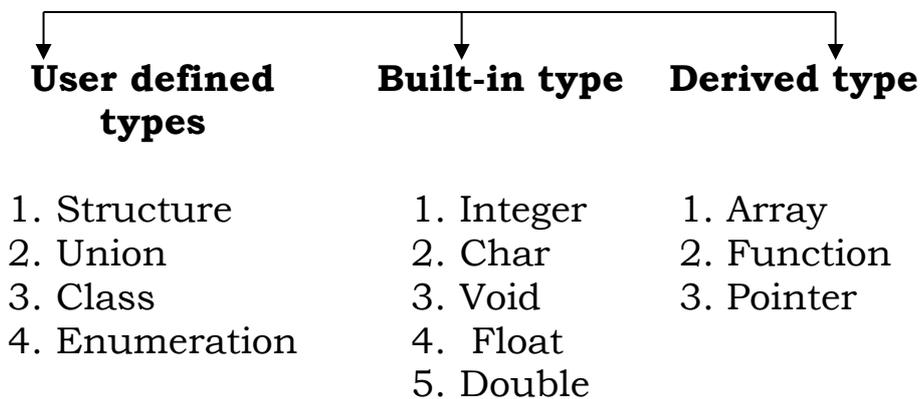
For example "COMPUTER" will be represented as "COMPUTER\0" in the memory and its size is 9 characters.

4. Data types

A data type is a type of data to hold in variable. It is a data storage format that can contain a specific type or range of values.

C++ support following types of data –

Data types



C++ has the following basic built-in data types -

- **Integer:** This data type is used to define integer numbers. It is a small integer number. The size of integer data type is 2 bytes.
For example `int count;`
`count=5;`
- **Float:** This data type is used to define floating point number. It is a small real number. The size of floating data type is 4 bytes.
For example `float miles;`
`Miles=5.2;`
- **Char:** This data type is used to define characters. It is used for single character. The size of char data type is 1 bytes.
For example `char letters;`
`letters='m';`
- **Double:** This data type is used big floating point numbers. It reserves twice the storage for the number. The size of double data type is 8 bytes.

For example double atoms;
atoms=2500000;

- **long double:** This data type is used for long floating point numbers. The size of long data type is 10 bytes.
For example long double population;
Population=10000000;
- **void:** The type void is used for a function, when function does not return any value.

5. Constants

A variable which does not change its value during execution of a program is known as a constant variable.

For example const float PI = 3.1415;
 const int RATE = 50;
 const float PI = 3.1415;
 const char CH = 'A';

6. Variables

A variable is a named area of storage that can hold a single value (numeric or character). Variable names are the symbolic representation of a memory location. The variables can be used to hold different values at different times during the execution of a program.

For example int a,b,c;
 int sum=0,f=1;

Rules for writing variable name

Variable name can be composed of letters (both uppercase and lowercase letters), digits and underscore '_' only.

- The first letter of a variable should be either a letter or an underscore.
- Spaces are not allowed.
- Keywords are not allowed as a variable name.

6. Operators

Operators are special symbols that used in C++ program to form an expression. Operator is used to transform one or more values into a single resultant value.

Operators are basically classified into three categories. These are -

▪ **Unary operator**

The operators that operate a single operand to form an expression are known as Unary operators. The operators like + (increment) operator, - (decrement) operator.

▪ **Binary operator**

The operators that operate two or more operands are known as Binary operators. The operators like +, -, *, /, % etc.

For example – a+b, a-b, a*b, a/b, a%b

▪ **Ternary operator**

The operator that operates minimum or maximum three operands is known as Ternary operator. There is only one ternary operator available in C++. The operator is ?: that is used as a substitute of if—else statement.

For example a>b ? a : b

Types of Binary operators

C++ language support following types of operators –

- Arithmetic operators
- Relational operators
- Logical operators
- Unary operators
- Assignment operators
- Increment and decrement operators
- Conditional operators
- The comma operator
- The size of operator
- The order of precedence
- Bitwise operator
- Scope resolution operator

▪ **Arithmetic operators**

Arithmetic operators are used to perform an arithmetic operation/expression. These operators are

+ (Addition), -(Subtraction), *(Multiplication), /(Division), %(Modulus)

▪ **Relational operators**

Relational operator also called Comparison operator. These operators are used to test the relation between two values. It requires two operands. A relational expression returns zero when the relation is false and a non-zero when it is true.

These operators are

Relational operators	Purpose
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to (equality)
!=	Not equal to

▪ **Logical operators**

The logical operators are used to combine one or more relational expression. The logical operators are

Logical operators	Purpose
&&	AND if both the condition are true then the result is true
	OR if one of the condition is true then the result is true
!	If both the condition are true then the result is false

▪ **Unary operators**

This provides two unary operators for which only one variable is required.

For Example

```
a = - 50;
```

```
a = + 50;
```

Here plus sign (+) and minus sign (-) are unary because they are not used between two variables.

▪ **Assignment operators**

The assignment operator '=' is used for assigning a variable to a value. This operator takes the expression on its right-hand-side and places it into the variable on its left-hand-side.

For example

```
m = 5;
```

It also support compound assignment operators.

Compound Assignment Operators

Operator	Example	Equivalent to
+=	A+=2	A=A+2
-=	A-=2	A=A-2
%=	A%=2	A=A%2
/=	A/=2	A=A/2
=	A=2	A=A*2

▪ **Increment and decrement operators**

C++ provides two special operators. These are '++' and '--' for incrementing and decrementing the value of a variable by 1. They can be used with any type of variable but it cannot be used with any constant. It has two forms, pre and post.

The syntax of the increment operator is:

Pre-increment: ++variable

Post-increment: variable++

The syntax of the decrement operator is:

Pre-decrement: `—variable`

Post-decrement: `variable—`

In Prefix form first variable is first incremented/decremented, then evaluated

In Postfix form first variable is first evaluated, then incremented/decremented

```
int x, y;
```

```
int i = 10, j = 10;
```

```
x = ++i; //add one to i, store the result back in x
```

```
y = j++; //store the value of j to y then add one to j
```

```
cout << x; //11
```

```
cout << y; //10
```

▪ **Conditional operators**

The conditional operator?: is called ternary operator. This requires three operands.

The general format of the conditional operator is

```
Conditional expression? expression1: expression2;
```

If the value of conditional expression is true then the expression1 is evaluated, otherwise expression2 is evaluated.

```
int a = 5, b = 6;
```

```
big = (a > b) ? a : b;
```

In the above statement, the condition evaluates to false, therefore biggest the value from b and it becomes 6.

▪ **The comma operator**

The comma operator gives left to right evaluation of expressions. The set of expressions has to be evaluated for a value, only the rightmost expression is considered.

For example `int a = 1, b = 2, c = 3, i;` // comma acts as separator, not as an operator

```
i = (a, b); // stores b into i
```

In the above statement first assign the value of a to i, and then assign value of b to variable i. So, at the end, variable i would contain the value 2.

▪ **The size of operator**

The sizeof operator can be used to find how many bytes are required for an object to store in memory.

```
For example          sizeof (char) returns 1
```

```
                    sizeof (float) returns 4
```

The sizeof operator determines the amount of memory required for an object at compile time rather than at run time.

When two operands of different type are encountered in the same expression, the lower type variable is converted to the higher type variable.

The following table shows the order of data types.

Order of data types

Data type	Order
Long double	
Double	Highest
Float	To
Int	lowest
Char	

In the above example, int value of b is converted to type float and stored in a temporary variable before being multiplied by the float variable c. The result is then converted to double so that it can be assigned to the double variable a.

2) Explicit conversion

It is also called type casting. It temporarily changes a variable data type from its declared data type to a new one. Here type casting can only be done on the right hand side the assignment statement.

```
totalPay = static_cast<double>(salary) + bonus;
```

in this example, initially variable salary is defined as float but for the above calculation it is first converted to double data type and then added to the variable bonus.

1.6 CONSOL INPUT / OUTPUT STATEMENT (I/O)

The following C++ stream objects can be used for the input/output purpose -

1) **cout** console

2) **cin** console

cout is output consol used to display message on screen with insertion (<<) operator. It inserts (or sends) the contents of the variable on its right to the object on its left.

For example

```
cout<< "Enter the value of num1";
```

```
cout<<250;
```

```
cout<<sum;
```

```
cout<<"Area of Rectangle="<<area;
```

cin is input consol used to input a value entered by the user from the keyboard with extraction (>>) operator.

For example

```
int marks;
```

```
cin >> marks;
```

1.7 STRUCTURE OF C++ PROGRAM

C++ program contains four sections. This is shown in figure –

Comment section / Documentations
Preprocessor directories / statements
Global declarations
Main() function { * Local declaration * Program statements and expression }

Here,

Comment section / Documentations

This is optional section. Comments are a way of explaining what makes a program. Comments are ignored by the compiler. There are two types of comments –

- a. Single line comments
- b. Multiline comments

Single line comments represents //.

For example // To print area of circle

Multiline comments represents /* */.

For example /* This program written by M. Kumar

To print factorial of any given number */

Processor directories/ Statements

C++ provides two preprocessor directives -

- 1) #include
- 2) #define

The #include is a preprocessor directive and to define header files such as iostream.h, conio.h, math.h, string.h, graphics.h etc.

For example,

```
#include<iostream.h>
```

```
#include<conio.h>
```

The #define is a preprocessor directive and to define symbolic name. Symbolic name is any valid variable names are written in upper case letters only.

For example,

```
#define PI 3.14
```

Here #define is the symbolic name

Main() function represents int main() / void main(). The main() is the main function where program execution begins. Every C++ program should contain only one main function.

Opening and closing Braces represents { and }. Two curly brackets are used to group all statements together.

For example,

```
/* This is C++ program*/ - Comment line
#include<iostream.h>      - Preprocessor directive
void main()              - main function
{
    Cout<<" C++ is a object oriented programming language";
}
```

Output

C++ is a object oriented programming language.

Example Write C++ program to calculate the area of rectangle

```
#include <iostream>
#include<conio.h>
void main ()
{
    int length , breadth , area;
    clrscr();
    cout << "Enter length of rectangle: ";
    cin >> length;
    cout << "Enter breadth of rectangle: ";
    cin >> breadth;
    area = length * breadth;
    cout << "Area of rectangle = " << area;
    getch();
}
```

Output

```
Please enter length of rectangle: 6
Please enter breadth of rectangle: 4
Area of rectangle is 24
```

1.8 CONTROL STRUCTURES

There are three types of control structures used in C++:

1. Sequence structure i.e. straight line structure
2. Selection structure i.e. branching/ Decision making / Conditional structure
3. Loop structure i.e. Iteration or repetition structure

1. Sequence structure

In this structure, statements are executed one after another i.e sequentially.

For example

```
#include<iostream.h>
void main()
{
float area,pi=3.14,r;
cout<<" Enter the value of radius :";
cin>>r;
area= pi * r * r;
cout<<"Area of circle ="<< area;
}
```

2. Selection structure

In this structure, one or more conditions to be evaluated along with statement or statements.

Types of Selection or Branching statements are as

- If statement
- If --- else statement
- If ---- else if ----- else statement
- Switch statement
- **If ---- statement**

The general format is

If (condition)

```
{
Statement/statements;
...
}
```

Here, if statement check the condition. If the condition is true then the result will display otherwise it does not display result.

For example

Write program in C++ to inputted any two given number and check it which is largest.

```
#include<iostream.h>
void main()
{
int num1,num2;
cout<<"Enter the num1";
cin>>num1;
cout<<"Enter the num2";
cin>>num2;
if(num1>num2)
{
Cout<<num1<<"is largest number";
}
}
```

Output

```
Enter the num1 34
Enter the num2 24
34 is largest number
```

• If ---- else statement

The general format is

If (condition)

```
{
Statement1;
}
else
{
Statement2;
}
```

Here, if statement checked the condition. If the condition is true then statement1 will be executed otherwise statement2 will be executed.

For example

Write program in C++ to inputted any two given number and check it which is largest.

```
#include<iostream.h>
void main()
{
int num1,num2;
cout<<"Enter the num1";
cin>>num1;
cout<<"Enter the num2";
```

```

cin>>num2;
if(num1>num2)
{
cout<<num1<<"is largest number";
}
else
{
cout<<num2<<"is largest number";
}

```

Output

```

Enter the num1 21
Enter the num2 24
24 is largest number

```

• If ---- elseif--- else statement

The general format is

```

If ( condition1 )
{
Statement1;
}
Else if( condition2 )
{
Statement2;
}
..
..
else
{
Statement n;
}

```

Here, this is also called nested statement. If statement check the condition1. If the condition1 is true then statement1 will be executed otherwise if the condition is false then the conditio2 will be checked if it is false then condition is checked. If all the conditions are false then else part will be executed.

For example

Write program in C++ to inputted any three given number and check it which is largest.

```
#include<iostream.h>
Void main()
{
int num1,num2,num3;
cout<<"Enter the num1";
cin>>num1;
cout<<"Enter the num2";
cin>>num2;
cout<<"Enter the num3";
cin>>num3;
if(num1>num2 && num1>num3)
{
cout<<num1<<"is largest number";
}
elseif(num2>num1 && num2>num3)
{
cout<<num2<<"is largest number";
}
else
{
cout<<num3<<"is largest";
}
}
```

Output

```
Enter the num1 34
Enter the num2 24
Enter the num3 45
45 is largest number
```

• Switch statement

This is a multiple branching statement, where based on a condition, the control is transfer to one of the many possible points.

The general format is

```
Switch (expression)
{
    Case1:
    {
        Action 1;
    }
    Case2:
    {
        Action 2;
    }
    :
    :
    Default :
    {
        Action n;
    }
}
```

The execution of switch statement begins with the evaluation of expression. If the value of expression matches with the constant then the statements following this statement execute sequentially till it executes break. The break statement transfers control to the end of the switch statement. If the value of expression does not match with any constant, the statement with default is executed.

For example

Write program in C++ to display the name of the day in a week, depending upon the number entered through the keyboard.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int day;
clrscr();
cout<<"Enter a number between 1 and 7\n";
cin>>day;
switch(day)
{
Case 1:
    Cout<<"Monday\n";
    Break;
Case 2:
    Cout<<"Tuesday\n";
    Break;
```

```

Case 3:
    Cout<<"Wednesday\n";
    Break;
Case 4:
    Cout<<"Thursday\n";
    Break;
Case 5:
    Cout<<"Friday\n";
    Break;
Case 6:
    Cout<<"Saturday\n";
    Break;
Case 7:
    Cout<<"Sunday\n";
Default:
    Cout<<"Invalid entry";
    Break;
}
getch();
}

```

Output

```

Enter a number between 1 to 7
2
Tuesday

```

2. Loop structure

In this structure, Loop is repeatedly executed statement or group of statements multiple times if the condition is true. If the condition is false then loop is terminated.

Types of looping statements are as

- While statement
- Do – while statement
- For statement

• While statement

The general format is

```

While(condition)
{
Statement/statements
}

```

Here, while checked the condition if statement or group of statements executed while given condition is true. If the condition is false then loop is automatically terminated.

For example -

Write program in C++ to print 1 to 10 numbers.

```
#include<iostream.h>
void main()
{
int i=1;
while(i<=10)
{
cout<<i<<"\n";
n=n+1;
}
```

• **do ---- while statement**

The general format is

```
do
{
Statement
}
```

While(condition);

Here the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

For example

Write program in C++ to print 1 to 10 numbers.

```
#include<iostream.h>
Void main()
{
int i=1;
do
{
Cout<<i<<"\n";
i=i+1;
}
While(i<=10);
}
```

- **For statement**

The general format is

```
for(initial value; test condition; increment)
{
Statement;
}
```

For loop has perform three operations -

1. Initialization of loop control variable
2. Testing of loop control variable
3. Update the loop control variable either by incrementing or decrementing.

Operation (1) is used to initialize the value. On the other hand, operation (2) is used to test whether the condition is true or false. If the condition is true, the program executes the body of the loop and then the value of loop control variable is updated. Again it checks the condition and so on. If the condition is false, it gets out of the loop.

For example -

Write program in C++ to print 1 to 10 numbers

```
#include<iostream.h>
Void main()
{
int i;
for(i=1;i<=10;i++)
{
Cout<<i<<"\n";
}
```

- **Nested for loop**

The general format is

```
for (initialization ; condition ; exp)
{
for(initialization ; condition ; exp)
{
Statements
}
}
```

When on one for statement is used within another for statement, then it is called nesting of for loops.

The first for loop is the outer loop. it is used for row and second for loop is inner loop. It is used for columns.

For example

```
#include<iostream.h>
#include<conio.h>
void main()
{
int i,j;
for(i=0;i<=5;i++)
{
for(j=1;j<=I;j++)
{
cout<<j;
}
cout<<"\n";
}
getch();
}
```

1.9 JUMP STATEMENT

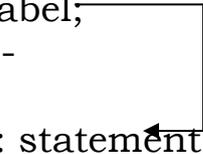
The jump statements unconditionally transfer program control within a function

1. Labels and goto statement
2. break statement
3. continue statement

• Labels and goto statement

The general format of goto and label statement are

```
goto label;
-----
-----
Label: statement
```



A goto statement is unconditional branching statement. A goto statement can cause program control to end up anywhere in the program checking for any condition.

A goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name and must be followed by colon. A label is placed immediately before the statement where the control is to be transferred.

For example

```
#include<iostream.h>
#include<conio.h>
void main()
{
int a,b;
cout<<"Enter the value of a and b=";
cin>>a>b;
if(a>b)
{
goto label1-para;
}
else
{
goto label2-para;
}
label1-para:
cout<<"a is largest number";
label2-para:
cout<<"b is largest number";
getch();
}
```

• The break statement

The general format is

```
break;
```

Here, break statement exit from loop. When break statement is encountered inside a loop, the loop is immediately existed and entire program continues with the statement immediately following the loop. When the loops are nested, the break would only exit from the loop containing it. That is, break will exit only a single loop.

The break statement can be used while loop, do—while loop and for loop.

For example

```
for( t=0 ; t<100 ; ++t)
{
count = 1;
for( ; ;)
{
cout<<count;
count++;
If(count==10)
```

```
Break;  
}  
}
```

- **The continue statement**

The general format is

```
continue;
```

Here, when continue is used inside any loop, control automatically passes to the beginning of the loop. when use continue within loop, continue causes the conditional test and increment portions of the loop to execute.

The continue statement skips rest of the loop body and starts a new iteration.

For example

```
for(i=0 ; i<4 ; i++)  
{  
cin>>number;  
if(number ==0)  
continue;  
k=1.0 / number;  
cout<<k;  
}
```

- **The exit statement;**

The general format is

```
exit ( code ) ;
```

Here, the execution of a program can be stopped at any point with exit and a status code can be informed to the calling program.

Where, code is an integer value. The code has a value 0 for correct execution. The value of the code varies depending upon the operating system.

For example

```
for(i=0;i<5;i++)  
{  
cout<<I;  
exit;  
}
```

1) Write program in C++ to check given number is Negative or Positive.

```
#include<iostream.h>
void main()
{
int num;
clrscr();
cout<<"Enter the number=";
cin>>num;
if(num>0)
{
Cout<<num<<"is Positive number";
}
Else
{
Cout<<num<"is Negative number";
}
getch();
}
```

Output

```
Enter the number = 4
4 is positive number
```

2) Write program in C++ to check given number is Odd or Even

```
#include<iostream.h>
void main()
{
int num;
clrscr();
cout<<"Enter the number=";
cin>>num;
if(num%2==0)
{
Cout<<num<<"is Even number";
}
Else
{
Cout<<num<"is Odd number";
}
getch();
}
```

Output

Enter the number = 4
4 is Even number

1.10 FUNCTIONS

A function is self contained block of statements that perform a specific task.

Use of Function

Functions are used to manipulate data item values and return a result.

Advantages of Functions

- 1) To reduce the length of program
- 2) To save memory space
- 3) Read, Write and debug program is easier
- 4) Modification and maintenance of program is easy

Types of Function

Functions are two types –

- 1) Library function
- 2) User defined function

Library function is also called pre-defined functions. These functions are in-built in Lib directory.

For example,

Max(), Min(), length(), abs(), sum(), sqrt(), pow() etc.

User defined function means user can defined own function.

The general format is

```
Return-type function-name(arg1,arg2,...)
```

```
{  
Function body  
}
```

Here, function name is valid name of identifier. Argument is user supplied values or variables.

For example,

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
int add(int x, int y)
```

```
{  
return(x+y);  
}
```

```
void main()
```

```
{  
int a,b;  
clrscr();
```

```

cout<<"Enter the value of A and B =";
cin>>a>>b;
add(a,b);
cout<<"Addition of A and B="<<(a*b);
getch();
}

```

1.11 FUNCTION PROTOTYPE

The prototype is a declaration that defines the arguments passed to the function and type of value returned by the function.

The general form is

Type function–name (argument list);

Here, the argument list contains the types and names of arguments that must be passed to the function.

For example – float volume (int x, float y, float z);

1.12 ARGUMENTS TO A FUNCTION

The calling function supplies some values to the called function. These are known as parameters. The variables which supply the values to a calling function called **actual parameters**. The variable which receive the value from called statement are termed **formal parameters**.

Consider an example of actual and formal parameters

```

#include<iostream.h>
void area(float)
void main()
{
    float radius;
    cin>>radius;
    area(radius);
    return 0;
}
void area(float r)
{
    Cout<<"The area of the circle is "<<3.14*r*r<<"\n"
}
getch();
}

```

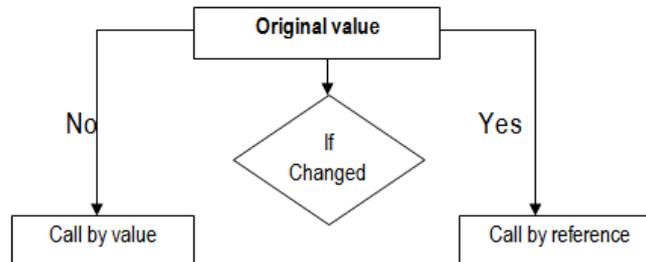
Here radius is called **actual parameter** and r is called **formal parameter**.

1.13 CALL BY VALUE AND CALL BY REFERENCE

There are two ways to pass data or argument to a function:

- 1) Call by value
- 2) Call by reference

Figure shows the operation of call by value and call by reference.



In call by value, original value cannot be changed.

In call by value, when a portion of the program invokes a function, control will be transferred from the main function to the calling function and the value of actual arguments is copied to the function. Within function the actual value may be altered or changed. When the control is transferred back from function to the program, altered values are not transferred back.

Consider an example of Call by value

```
#include<iostream.h>
#include<conio.h>
Void swap(int a, int b)
{
Int temp=a;
a=b;
b=temp;
}
void main()
{
Int a=100, b=200;
Swap(a,b);
Cout<<"Value of a="<<a;
Cout<<"Value of b="<<b;
Getch();
}
```

Output

Value of a=100
Value of b=200

In call by reference, original values can be changed.

In call by reference, when a function is called by a program the address of the actual arguments are copied onto the formal

arguments i.e. the formal and actual arguments are referencing to same memory location. Therefore change in value of formal argument affects the value of actual arguments. The call by reference is used when function produces more than one value and provides these values to the caller.

Consider an example of Call by reference

```
#include<iostream.h>
#include<conio.h>
Void swap(int *a, int *b)
{
Int temp=*a;
*a=*b;
*b=temp;
}
void main()
{
Int *a=100, *b=200;
Swap(&a,&b);
Cout<<"Value of a="<<a;
Cout<<"Value of b="<<b;
getch();
}
```

Output

```
Value of a=200
Value of b=100
```

1.14 RETURN STATEMENT

A return statement may send value to the calling function. A return statement can return only one value per call.

It can have several forms –

- 1) return;
- 2) return(expression);

In the first form of return, return does not return any value but it only returns the control of execution to the calling function.

In the second form of return, returns values of expression.

For example,

```
#include<iostream.h>
#include<conio.h>
Int add(int x, int y)
{
return(x+y);
}
```

```

Void main()
{
Int a,b;
Cout<<"Enter two number="";
Cin>>a>>b;
Add(a,b);
Cout<<"Addition =">>(a+b);
}

```

1.15 INLINE FUNCTION

An inline function to put code in function body directly inside the program. So function is called. Such function is called inline function.

Inline function is a function that is expanded in line when it is invoked i.e. compiler replaces the function call with corresponding function code. It is useful for small size functions. Inline function defined with inline keyword.

To save execution time in short functions, inline function is used.

The general form of inline function is

inline function-header

```

{
    Function body
}

```

Rules for use of inline function

- Function is made inline by putting a word inline in the beginning.
- Inline function should be declared before main() function.
- It does not have function prototype.
- Only shorter code is used in inline function.

Consider an example of inline function

```

#include<iostream.h>
#include<conio.h>
inline int multi(int x, int y)
{
return(x*y);
}
inline int add(int x,int y)
{
return(x+y);
}
void main()
{
int i,j;

```

```

clrscr();
cout<<"Enter i and j";
cin>>i>>j;
cout<<"Add"<<add(i,j)<<"\n";
cout<<"multi"<<multi(i,j);
getch();
}

```

Output

```

Enter I and j 3 4
Add 7
Multi 12

```

1.16 DEFAULT AND CONST ARGUMENTS

A default argument is a value provided in function declaration that is automatically assigned by the compiler if caller of the function doesn't provide a value for the argument with default value.

Consider an example of default arguments

```

/* function with default arguments, it can be called with
2 arguments or 3 arguments or 4 arguments.*/
#include<iostream.h>
int sum(int x, int y, int z=0, int w=0)
{
    return (x + y + z + w);
}
/* program to test above function*/
int main()
{
    cout << sum(10, 15) << endl;
    cout << sum(10, 15, 25) << endl;
    cout << sum(10, 15, 25, 30) << endl;
    return 0;
}

```

Output

```

25
50
80

```

1.17 FUNCTION OVERLOADING

Overloading means the use of same thing for different purposes. Function overloading means the use of same function name to create functions that perform a variety of different tasks. This is known as function polymorphism.

Consider an example of function overloading,

```
#include<iostream.h>
#include<conio.h>
void test(int);
void test(float);
void test(int, float);
void main()
{
int a=5;
float b=5.5;
clrscr();
test(a);
test(b);
test(a,b);
getch();
}
void test(int n)
{
cout<<"Integer number:"<<n<<"\n";
}
void test(float n)
{
cout<<"Float number:"<<n<<"\n";
}
void test(int n1, float n2)
{
cout<<"Integer :"<<n1<<"\n";
cout<<"Float :"<<n2;
}
}
```

Output

Printing Addition: 12

Printing Subtraction: 283.56

1.18 ARRAY

An array is a collection of homogenous data elements of same data type. It is described by a single name and each element of an array is referenced by using array name and its subscript no.

There are two types of array –

1. One dimensional array
2. Two dimensional array

1. One dimensional array

Declaration of 1D Array

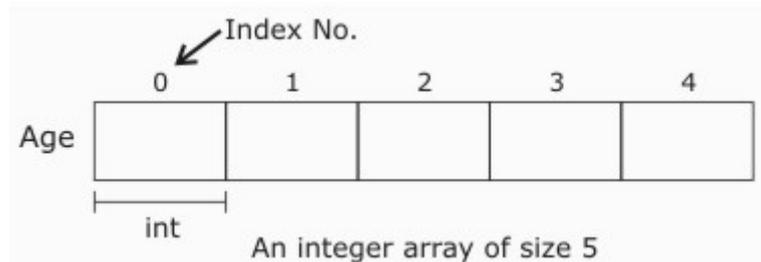
The general format is

```
DataType arrayName[size of array];
```

For example,

```
int age[5];
```

```
float cost[30];
```



Initialization of One Dimensional Array

An array can be initialized along with declaration. For array initialization it is required to place the elements separated by commas enclosed within braces.

```
int a[5]={11,22,23,4,15};
```

It is possible to leave the array size open. The compiler will count the array size.

```
int b[]={6,7,8,9,15,12};
```

Referring to Array Elements

In any point of a program in which an array is visible, it can access the value of any of its elements individually as if it was a normal variable, thus being able to both read and modify its value.

The format is as simple as:

```
name[index]
```

For example

```
cout<<age[4];           // print an array element
```

```
age[4]=55;              // assign an array element
```

```
cin>>age[4];           // input element 4
```

Example

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
Void main()
```

```
{
```

```
Int a[n];
```

```
Cout<<"Enter the length of array";
```

```
Cin>>n;
```

```
For(int i=0;i<n;i++)
```

```
{
```

```
Cin>>a[i];
```

```

}
Cout<<"To print array elements";
For(i=0;i<n;i++)
{
Cout<<a[i]<<"\n";
}
}
}

```

2. Two Dimensional Array

It is a collection of data elements of same data type arranged in rows and columns (that is, in two dimensions).

Declaration of Two-Dimensional Array

The general format is

```
DataType arrayName[number Of Rows] [number Of Columns];
```

For example

```
int sales[3][5];
```

Initialization of Two-Dimensional Array

An two-dimensional array can be initialized along with declaration. For two-dimensional array initialization, elements of each row are enclosed within curly braces and separated by commas. All rows are enclosed within curly braces.

For example

```
int a[4][3]={{22,23,10},{15,25,13},{20,74,67},{11,18,14}};
```

Referring to Array Elements

To access the elements of a two-dimensional array, need a pair of indices: one for the row position and one for the column position.

The format is as

```
name [ row_Index ] [ column_Index ]
```

for examples:

```
cout<<a[1][2];           // print an array element
a[1][2]=13;             // assign value to an element
cin>>a[1][2];           // input element
```

Example

```
#include<iostream.h>
```

```
#include<conio.h>
```

```
Void main()
```

```
{
```

```
Int a[r][c];
```

```
Cout<<"Enter the number of rows and columns of matrix";
```

```
Cin>>r>>c;
```

```
For(int i=0;i<r;i++)
```

```
{
```

```

For(int j=0;j<c;j++)
{
Cin>>a[r][c];
}
}
Cout<<"To print array elements";
For(i=0;i<r;i++)
{
For(j=0;j<c;j++)
{
Cout<<a[r][c]<<"\n";
}
}
}

```

1.19 MANIPULATORS

Manipulators are operators that are used to format the data display. The most commonly used manipulators are endl and setw. The endl operator is similar to that of '\n'. It causes a line feed to be inserted.

For example

```

cout<< "m=" <<m << endl;
cout<< "n=" <<n << endl;

```

This would cause two lines of output.

```

m = 1 2 3 4
n = 12

```

The manipulator setw specifies fieldwidth.

For example

```

cout<< setw (5)<< sum << endl;

```

Here the field width is 5 for printing the value of variable sum.

1.20 ENUMERATION

Enumeration is a user defined data type that consists of integer constants. To define an enumeration, keyword enum is used.

The general format is

```

enum enum_name{arg1, arg2, .....}

```

For example: enum color{yellow, green, black, white}

Here,

the name of the enumeration is color and yellow, green, black and white are values of type color. By default yellow is 0, green is 1, black is 2 and white is 3. Also it can change the default value of an enum element during declaration.

Example

```

#include<iostream.h>

```

```
#include<conio.h>
Enum color{red,green,blue};
Void main()
{
Color code1,code2,cod3;
Code1=red;
Code2=green;
Code3=blue;
Cout<<"Color code="<<code1;
Cout<<"Color code="<<code2;
Cout<<"Color code="<<code3;
}
```

Output

Color code1=0

Color code2=1

Color code3=2

Note: To change the default value of enum elements - **code=green+1**

-----*-----*-----*-----*-----*-----

Assignment Unit – 1

1. Explain data types in C++
2. Explain if--- else if statement with suitable example.
3. Define inline function with example.
4. What is meant by default and constant arguments.
5. Write note on manipulators.

Assignment Web site :

- 1) <http://www.sanfoundry.com/c-plus-plus-interview-questions-and-answers-arrays/>**
- 2) www.cppforschool.com/assignm (More examples in C++)**

Assignment Questions Unit – I

Q. 1 Select the correct answer

1) The value 132.54 can be represented using which data type?

- a) double
- b) void
- c) int
- d) bool

Ans :-

2) What is the size of long data type?

- a) 2 byte
- b) 10 byte
- c) 8 byte
- d) 4 byte

Ans :-

3) Which of the following correctly declares an array?

- a) int array[10];
- b) int array;
- c) array{10};
- d) array array[10];

Ans:-

4) Which of the following gives the memory address of the first element in array?

- a) array[0];
- b) array[1];
- c) array(2);
- d) array;

Ans:-

Q.2 What will be the output of the this program?

```
#include <iostream.h>
#include<conio.h>
void main ()
{
    int array[] = {0, 2, 4, 6, 7, 5, 3};
    int n, result = 0;
    for (n = 0; n < 8; n++) {
        {
            result += array[n];
        }
    }
    cout << result;
}
```

Q.3 What is the output of this program?

```
#include<iostream.h>
void main()
{
    int a = 5, b = 10, c = 15;
    int arr[3] = {&a, &b, &c};
    cout << *arr[*arr[1] - 8];
}
```

Q.4 What is the output of this program?

```
#include <conio.h>
#include<iostream.h>
void main()
{
    char str[5] = "ABC";
    cout << str[3];
    cout << str;
}
```

Q.5 Re-arrange the code

1. void main()
2. {
3. add.get();
4. add.show();
- 5.add.cal();
6. op add;
7. }
8. void get()
- 9.{
10. cout<<"Enter n1 and n2 value=";
11. cin>>n1>>n2;
- 12.}
13. class op
14. {
15. private:
16. int n1,n2,n3;
17. public:
18. void cal()
19. {
20. n3=n1+n2;
- 21.}
22. void show()
- 23.{

```
24. cout<<"Addition="<<n3;  
25.}  
26.);
```