Unit-I
Object Oriented Programming in C++

Revision Tour of Class XI Chapter: 01

➢ **Keywords:** Keywords are the certain reserved words that convey a special meaning to the compiler. These are reserve for special purpose and must not be used as identifier name.eg for , if, else , this , do, etc.

➢ **Identifiers:** Identifiers are programmer defined names given to the various program elements such as variables, functions, arrays, objects, classes, etc.. It may contain digits, letters and underscore, and must begin with a letter or underscore. C++ is case sensitive as it treats upper and lower case letters differently. A keyword can not be used as an identifiers. The following are some valid identifiers:

Pen time580 s2e2r3 _dos _HJI3_JK

➢ **DataTypes in C++:** Data types are means to identify the types of data and associated operations of handling it. Data types in C++ are of two types:

1. Fundamental or Built-in data types: These data types are already known to compiler. These are the data types which are not composed of other data types. There are following fundamental data types in C++:

(i) int data type (for integer) (ii) char data type (for characters)
(iii) float data type (for floating point numbers) (iv) double data type

2. Derived and User defined date types : These data types are made up of fundamental data types **:** For example 1)Array 2) Function 3) Reference 4) Constant 5) Pointer 6) Class 7) Enum 8) Union 9) Structure

Data Type Modifiers: There are following four data type modifiers in C++ , which may be used to modify the fundamental data types to fit various situations more precisely:

(i) signed (ii) unsigned (iii) long (iv) short

➢ **Variables:** A named memory location, whose contents can be changed with in program execution is known as variable. OR

A variable is an identifier that denotes a storage location, whose contents can be varied during program execution.

Declaration of Variables: Syntax for variable declaration is:

datatype variable_name1, variable_name2, variable_name3,…… ;

We can also initialize a variable at the time of declaration by using following syntax:

datatype variable_name = value;

When the initial value is given to the variable at the run time it is called dynamic initialization.e.g.

```
float avg;
avg = sum/count;
```

then above two statements can be combined in to one as follows:

```
float avg = sum/count;
```

➢ **Constant:** A named memory location, whose contents cannot be changed within program execution is known as constant. OR

A constant is an identifier that denotes a storage location, whose contents cannot be varied during program execution.

Syntax for constant declaration is:

```
const datatype constant_name = value ;
```

e.g., const float pi = 3.14f ;

➢ Conditional operator (? :):

The conditional operator (? :) is a ternary operator i.e., it require three operands. The general form of conditional operator is:

expression1? expression2: expression3 ;

Where expression1 is a logical expression , which is either true or false.
If expression1 evaluates to true i.e., 1, then the value of whole expression is the value of expression2, otherwise, the value of the whole expression is the value of expression3. For example
min = a<b? a : b ;
Here if expression (a<b) is true then the value of a will be assigned to min otherwise value of b will be assigned to min.

➢ Type Conversion:

The process of converting one predefined data type into another is called type conversion.

C++ facilitates the type conversion in two forms:

(i) **Implicit type conversion**:- An implicit type conversion is a conversion performed by the compiler without programmer's intervention. An implicit conversion is applied generally whenever different data types are intermixed in an expression. The C++ compiler converts all operands upto the data type of the largest data type's operand, which is called type promotion.

(ii) **Explicit type conversion** :- An explicit type conversion is user-defined that forces an expression to be of specific data type.

Type Casting:- The explicit conversion of an operand to a specific type is called type casting.

Type Casting Operator - (type) :-Type casting operators allow you to convert a data item of a given type to another data type. To do so , the expression or identifier must be preceded by the name of the desired data type , enclosed in parentheses . i. e.,

(data type) expression

Where data type is a valid C++ data type to which the conversion is to be done. For example , to make sure that the expression (x+y/2) evaluates to type float , write it as:

(float) (x+y/2)

➢ Some important Syntax in C++:

1. if Statement

```
if ( < conditional expression > )
{
< statement-1 or block-1>;
   // statements to be executed when conditional expression is true.                    }
[  else
{
< statement-2 or block-2>;
   // statements to be executed when conditional expression is false.
}   ]
```

2. **The if-else-if ladder :**

```
if ( <condition -1> )
        statement-1;        // do something if condition-1 is satisfied (True)
    else if ( <condition – 2 >)
        statement-3 ;    // do something if condition -2 is satisfied (True)
    else if (<condition – 3 >)
        statement-3 ;        // do something if condition- 3 is satisfied (True)
                :
```

```
                                        :    // many more n-1 else - if ladder may come
                                        :
                    else if( < condition – n >)
                            statement-n ;          // do something if condition – n is satisfied (True)
            [       else
                            statement-m ;    ]    // at last do here something when none of the
                                                  // above  conditions gets satisfied (True)
                    }
```

<>	in syntax is known as a place holder, it is not a part of syntax, do not type it while writing program. It only signifies that anything being kept there varies from program to program.
[ ]	is also not a part of syntax , it is used to mark optional part of syntax i.e. all part of syntax between [ ] is optional.

3. switch Statement :-

```
        switch (expression/variable)
{       case value_1:  statement -1;
                       break;
                case value_2:  statement -2;
                       break;
                :
                :
                case value_n:  statement -n;
                       break;
                [  default:  statement -m;   ]
}
```

4. The for Loop:

```
for(initialization_expression(s); loop_Condition; update_expression (s))
                 {
                 Body of loop
                 }
```

5. while Loop:

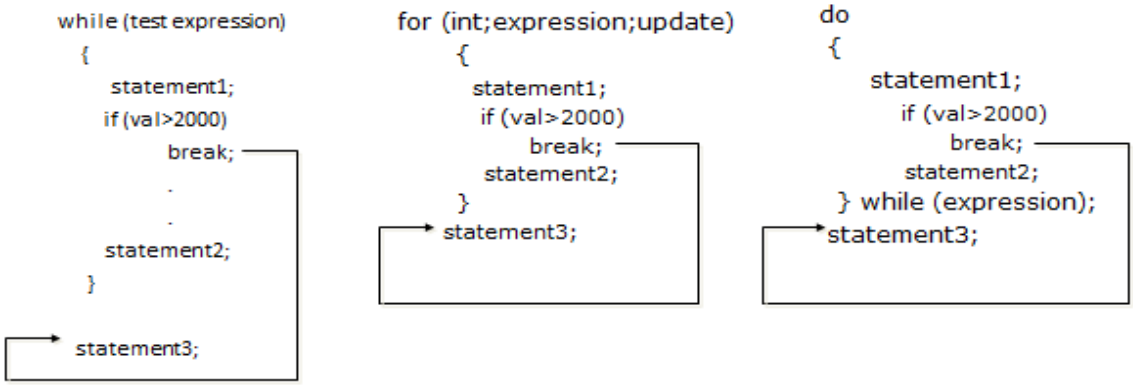
```
        while (loop_condition)
                {
                        Loop_body
                }
```

6. do-while loop:

```
                        do
                          {             Loop_body
                        }while (loop_condition);
```

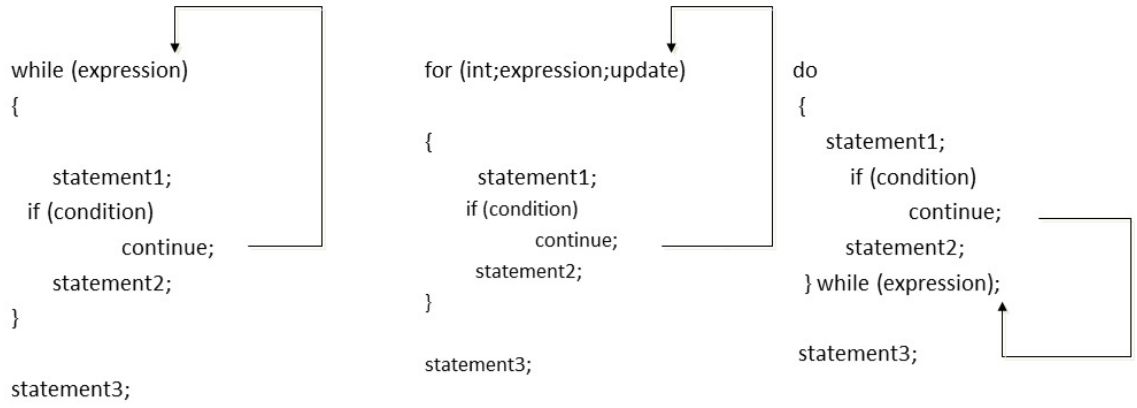
- **break Statement :-**The break statement enables a program to skip over part of the code. A break statement terminates the smallest enclosing while, do-while, for or switch statement. Execution resumes at the statement immediately following the body of the terminated statement. The following figure explains the working of break statement:

The Working of Break Statement



➢ **continue Statement:-** The continue is another jump statement like the break statement as both the statements skip over a part of the code. But the continue statement is somewhat different from break. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between. The following figure explains the working of continue statement:

The Working of Continue Statement



➢ **Functions :-**Function is a named group of programming statements which perform a specific task and return a value.

1. **Built-in Functions (Library Functions)** :- The functions, which are already defined in C++ Library (in any header files) and a user can directly use these function without giving their definition is known as built-in or library functions. e.g., sqrt( ), toupper( ), isdigit( ) etc.

Following are some important Header files and useful functions within them :

stdio.h (standard I/O function)	gets( ) , puts( )
ctype.h (character type function)	isalnum( ) , isalpha( ), isdigit ( ), islower (), isupper ( ), tolower ( ), toupper( )
string.h (string related function)	strcpy ( ), strcat ( ), strlen( ), strcmp( ) , strcmpi( ) , strrev( ),strupr( ) , strlwr( )
math.h (mathematical function)	fabs ( ), pow ( ), sqrt ( ), sin ( ), cos ( ), abs ( )
stdlib.h	randomize ( ), random ( )

randomize( ) : This function provides the seed value and an algorithm to help random( ) function in generating random numbers. The seed value may be taken from current system's time.

random(<int>) : This function accepts an integer parameter say x and then generates a random value between 0 to x-1.

for example : random(7) will generate numbers between 0 to 6.

To generate random numbers between a lower and upper limit we can use following formula

random(U – L +1) + L

where U and L are the Upper limit and Lower limit values between which we want to find out random values.

For example : If we want to find random numbers between 10 to 100 then we have to write code as:

random(100 -10 +1) + 10 ; // generates random number between 10 to 100

2. **User-defined function :-** The functions which are defined by user for a specific purpose is known as user-defined function. For using a user-defined function it is required, first define it and then using.

Declaration of user-defined Function:

```
Return_type  function_name(List of formal parameters)
  {
  Body of the function
}
```

Calling a Function:- When a function is called then a list of actual parameters is supplied that should match with formal parameter list in number, type and order of arguments.

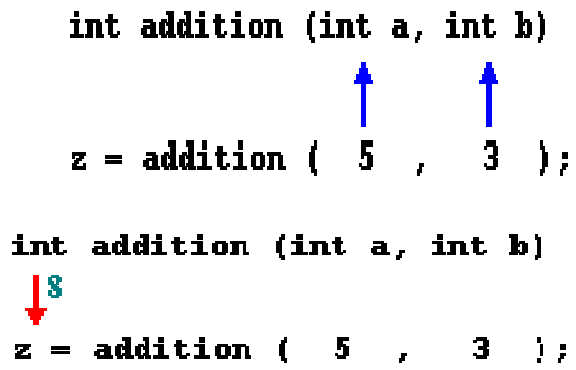
Syntax for calling a function is:

function_name (list of actual parameters);

e.g.,

```
#include <iostream.h>
 int addition (int a, int b)
{  int r;
   r=a+b;
return (r); }
void main ( )
{  int z ;
   z = addition (5,3);
cout<< "The result is " << z;
  }
```

The result is 8



Call by Value (Passing by value) :- The call by value method of passing arguments to a function copies the value of actual parameters into the formal parameters , that is, the function creates its own copy of argument values and then use them, hence any chance made in the parameters in function will not reflect on actual parameters . The above given program is an example of call by value.

Call by Reference (Passing by Reference) :- The call by reference method uses a different mechanism. In place of passing value to the function being called , a reference to the original variable is passed . This means that in call by reference method, the called function does not create its own copy of original values , rather, its refers to the original values only by different names i.e., reference . Thus the called function works on the original data and any changes are reflected to the original values.

// passing parameters by reference

```
#include <iostream.h>
        void duplicate (int& a, int& b, int& c)
```

```
{
        a*=2;
        b*=2;
        c*=2;
}

void main ()
        {
                int x=1, y=3, z=7;
                duplicate (x, y, z);
                cout <<"x="<< x <<", y="<< y <<", z="<< z;
        }
```

```
void duplicate (int& a,int& b,int& c)
                 x      y      z
duplicate (   x  ,   y  ,   z  );
```

output :x=2, y=6, z=14

The ampersand (&)operator specifies that their corresponding arguments are to be passed *by reference* instead of *by value*.

Constant Arguments:-In C++ the value of constant argument cannot be changed by the function. To make an argument constant to a function , we can use the keyword const as shown below:

```
int myFunction( const int x , const int  b );
```

The qualifier const tell the compiler that the function should not modify the argument. The compiler will generate an error when this condition is violated.

Default Arguments :- C++ allows us to assign default value(s) to a function's parameter(s) which is useful in case a matching argument is not passed in the function call statement. The default values are specified at the time of function definition from right most parameter to left ones. e.g.,

```
float interest ( float principal, int time, float rate = 0.70f)
```

Here if we call this function as:

```
si_int= interest(5600,4);
```

then rate =0.7 will be used in function.

Formal Parameters:- The parameters that appear in function definition are formal parameters.

Actual Parameters :- The parameters that appears in a function call statement are actual parameters.

Functions with no return type (The use of void):- In this case we should use the void type specifier for the function. This is a special specifier that indicates absence of type.

The return Statement :- The execution of return statement, it immediately exit from the function and control passes back to the calling function (or, in case of the main( ), transfer control back to the operating system). The return statement also returns a value to the calling function. The syntax of return statement is:

```
return ( value);
```

- **Scope of Identifier** :-The part of program in which an identifier can be accessed is known as scope of that identifier. There are four kinds of scopes in C++

(i) **Local Scope** :- An identifier declare in a block ({ }) is local to that block and can be used only in it.

(ii) **Function Scope** :- The identifier declare in the outermost block of a function or in its argument list, have function scope.

(iii) **File Scope (Global Scope)** :- An identifier has file scope or global scope if it is declared outside all blocks i.e., it can be used in all blocks and functions.

(iv) **Class Scope :**- A name of the class member has class scope and is local to its class.

- **Lifetime :**The time interval for which a particular identifier or data value lives in the memory is called Lifetime of the identifier or data value.

➢ Arrays:

Declaration of One-Dimentional Array:-

```
Data_type  Array_name[size];
```

Working with One Dimentional Array:-

General form of for loop for Reading elements of array (1-D)	Generally processing part may be include within the loop of reading or printing, otherwise a same type separate loop may be used for processing	General form of for loop for printing elements of array (1-D)

for (int i=0; i< size; i++)
{
cout<<"Enter Array Element "<<i+1;
cin>>Array_Name[i];
}
		for (int i=0; i< size; i++)
		{
		cout<<Array_Name[i]<< ", ";
		}

Declaration of 2-D array:-

```
Data_type  Array_name [R][C] ;
```

Where R represent number of rows and C represent number of columns in array.

Working With Two-Dimentional Array:-

General form of for loop for Reading elements of 2-D array	Generally processing part may be include within the loop of reading or printing, otherwise a same type separate nested loop may be used for processing	General form of for loop for printing elements of 2-D array

for (int i=0; i< R; i++)
{ cout<<"Enter Row "<<i+1;
for (int j=0; j<C ; j++)
cin>>Array_Name[i][j];
}
		for (int i=0; i< R; i++)
		{ for (int j=0; j<C ; j++)
		cout<<Array_Name[i][j]<<'\t';
		cout<<'\n';
		}

Where R represent number of rows and C represent number of columns in array.

➢ Defining Structure :-

```
struct< Name of Structure >
        {
        <datatype>< data-member 1>;
        <datatype>< data-member 2>;
        <datatype>< data-member 3>;
                        …
                        …
        <datatype>< data-member n>;
        } ;
```

Declaring Structure Variable :-

```
        struct< Name of Structure >
        {
        <datatype>< data-member 1>;
        <datatype>< data-member 2>;
```

<datatype>< data-member 3>;
…
…
<datatype>< data-member n>;
} var1, var2,….., varn ;

We can declare the structure type variables separately (after defining of structure) using following syntax:

Structure_name var1, var2, …… ….., var_n;

Accessing Structure Elements :- To access structure element , dot operator is used. It is denoted by (.). The general form of accessing structure element is :
Structure_Variable_Name.element_name

- **Pointer:-** Pointer is a variable that holds a memory address of another variable of same type.

Declaration and Initialization of Pointers :
Syntax :
Datatype *variable_name;
e.g., int *p; float *p1; char *c;
Two special unary operator * and & are used with pointers. The & is a unary operator that returns the memory address of its operand.
e.g., int a = 10; int *p; p = &a;
Pointer arithmetic: Two arithmetic operations, addition and subtraction, may be performed on pointers. When you add 1 to a pointer, you are actually adding the size of whatever the pointer is pointing at. That is, each time a pointer is incremented by 1, it points to the memory location of the next element of its base type.
e.g. int *p; p++;
If current address of p is 1000, then p++ statement will increase p to 1002, not 1001.
Adding 1 to a pointer actually adds the size of pointer's base type.
Base address : A pointer holds the address of the very first memory location of array where it is pointing to. The address of the first memory location of array is known as BASE ADDRESS.
Dynamic Allocation Operators :C++ dynamic allocation operatorsallocate memory from the free store/heap/pool, the pool of unallocated heap memory provided to the program. C++ defines two operators **new** and **delete** that perform the task of allocating and freeing memory during runtime.
Pointers and Arrays :C++ treats the name of an array as constant pointer which contains base address i.e address of first memory location of array.

- **typedef :-** The typedef keyword allows to create alias(alternate name) for data types. the syntax is:

typedef existing_data_type new_name ;
e.g. typedef int num;

- **#define Preprocessor Directive:**The # define directive creates symbolic constant, constants that are represent as macros.

Macros: Macros are preprocessor directive created using # define that serve as symbolic constants. They are created to simplify and reduce the amount of repetitive coding
e.g.1
#define PI 3.14
Here PI is defined as a macro. It will replace 3.14 in place of PI throughout the program.
e.g. 2
#define max (a, b) a>b? a: b
Defines the macro max, taking two arguments a and b. This macro may be called like any

function. Therefore, after preprocessing
A = max(x, y);
Becomes A = x>y?x :y ;

- **Function Overloading:** Function overloading is the process of defining and using functions with same name having different argument list and/or different return types. These functions are differentiated during the calling process by the number, order and types of arguments passed to these functions.

  Example:
  int Add (int ,int) ;
  double Add (double ,double) ;
  float Add (int ,float) ;

Short Answer Type Questions (2-Marks)

1. Define Macro with suitable example.
2. Explain in brief the purpose of function prototype with the help of a suitable example.
3. What is the role of typedef? Can it be used to create new data type?
4. What is the difference between Object Oriented Programming and Procedural Programming?
5. What is the difference between Global Variable and Local Variable? Also, give a suitable C++ code to illustrate both.
6. Differentiate between ordinary function and member functions in C++. Explain with an example.
7. What is the difference between call by reference and call by value with respect to memory allocation? Give a suitable example to illustrate using C++ code.
8. What is the difference between actual and formal parameter ? Give a suitable example to illustrate using a C++ code.
9. Differentiate between a Logical Error and Syntax Error. Also give suitable examples of each in C++.
10. Find the correct identifiers out of the following, which can be used for naming variable, constants or functions in a C++ program :
    While, for, Float, new, 2ndName, A%B, Amount2, _Counter
11. Out of the following, find those identifiers, which cannot be used for naming Variable, Constants or Functions in a C++ program :
    _Cost, Price*Qty, float, Switch, Address One, Delete, Number12, do
12. Find the correct identifiers out of the following, which can be used for naming Variable, Constants or Functions in a C++ program :
    For, while, INT, NeW, delete, 1stName, Add+Subtract, name1

Very Short Answer Type Questions (1-Mark Based on Header Files)

1. Which C++ header file (s) will be included to run /execute the following C++ code?

```
void main( )
{ int Last =26.5698742658;
cout<<setw(5)<<setprecision(9)<<Last; }
```

   Ans: iostream.h, iomanip.h

2. Name the header files that shall be needed for successful compilation of the following C++ code :

```
void main()
{    char str[20],str[20];
```

```
gets(str);
strcpy(str1,str);
strrev(str);
puts(str);
puts(str1);      }
```

3. Write the names of the header files to which the following belong:
   (i) strcmp() (ii) fabs()
4. Write the names of the header files to which the following belong:
   (i) frexp() (ii) isalnum()

Short Answer Type Questions (2-MarksError Finding)

1. Rewrite the following program after removing any syntactical errors. Underline each correction made.

```
#include<iostream.h>
void main( )
int A[10];
A=[3,2,5,4,7,9,10];
for( p = 0; p<=6; p++)
{ if(A[p]%2=0)
int S = S+A[p]; }
cout<<S;
 }
```

Ans :-
```
#include<iostream.h>
void main( )
{int A[10] = {3,2,5,4,7,9,10};
int S = 0,p;
for(p = 0; p<=6; p++)
{ if(A[p]%2==0)
S = S+A[p]; }
cout<<S;}
```

2. Deepa has just started working as a programmer in STAR SOFTWARE company. In the company she has got her first assignment to be done using a C++ function to find the smallest number out of a given set of numbers stored in a one-dimensional array. But she has committed some logical mistakes while writing the code and is not getting the desired result. Rewrite the correct code underlining the corrections done. Do not add any additional statements in the corrected code

```
int find(int a[],int n)
{ int s=a[0];
  for(int x=1;x<n;x++)
 if(a[x]>s)
 a[x]=s;
return(s);  }
```

3. Rewrite the following program after removing the syntactical errors (if any). Underline each correction.

```
#include [iostream.h]
class PAYITNOW
{        int Charge;
PUBLIC:
   void Raise(){cin>>Charge;}
  void Show{cout<<Charge;}
};
void main()
{
PAYITNOW P;
P.Raise();
Show();
}
```

4. Rewrite the following program after removing the syntactical errors (if any). Underline each correction.

```
#include <iostream.h>
struct Pixels
{       int Color,Style;}
void ShowPoint(Pixels P)
{       cout<<P.Color,P.Style<<endl;}
void main()
{
        Pixels Point1=(5,3);
        ShowPoint(Point1);
        Pixels Point2=Point1;
        Color.Point1+=2;
        ShowPoint(Point2);
}
```

Short Answer Type Questions (2-MarksFinding Output)

2. Find the output of the following C++ program:

```
#include<iostream.h>
void repch(char s[])
{
for (int i=0;s[i]!='\0';i++)
{
if((i%2)!=0) &&(s[i]!=s[i+1]))
{
s[i]='@';
}
else if (s[i]==s[i+1])
{
s[i+1]='!';
}       }       }
void main()
{
char str[]="SUCCESS";
cout<<"Original String"<<str;
repch(str);
cout<<"Changed String"<<str;}
```

Ans: Original String SUCCESS
Changed String S@C@ES!

1. Find output of the following program segment :

```
#include<iostream.h>
#include<ctype.h>
void Mycode(char Msg[],char CH)
{  for(int cnt=0;Msg[cnt]!='\0';cnt++)
{ if(Msg[cnt]>='B'&& Msg[cnt]<='G')
Msg[cnt]=tolower(Msg[cnt]);
else
if(Msg[cnt]=='N'||Msg[cnt]=='n'||Msg[cnt]==' ')
Msg[cnt]=CH;
else
if(cnt%2==0)
Msg[cnt]=toupper(Msg[cnt]);
else
Msg[cnt]=Msg[cnt–1];  } }
void main()
{ char MyText[]="Input Raw";
Mycode(MyText,'@');
cout<<"NEW TEXT:"<<MyText<<endl;
}
```

3. Find the output of the following program:

```
#include <iostream.h>
#include <ctype.h>
void Encrypt(char T[])
{
for (int i=0;T[i]!='\0';i+=2)
        if (T[i]=='A' || T[i]=='E') T[i]='#';
        else if (islower(T[i])) T[i]=toupper(T[i]);
        else T[i]='@';
}
void main()
{
char Text[]="SaVE EArtH";
//The two words in the string Text
//are separated by single space
Encrypt(Text);
cout<<Text<<endl;
}
```

4. Find the output of the following program:

```
#include <iostream.h>
struct Game
{
char Magic[20];
int Score;
};
void main()
{
        Game M={"Tiger",500};
        char *Choice;
        Choice=M.Magic;
        Choice[4]='P';
        Choice[2]='L';
        M.Score+=50;
        cout<<M.Magic<<M.Score<<endl;
        Game N=M;
        N.Magic[0]='A';N.Magic[3]='J';
        N.Score-=120;
        cout<<N.Magic<<N.Score<<endl;
}
```

Application Based Questions (3 Marks Finding Output)

1. Find the output of the following :

```
#include<iostream.h>
void switchover(int A[ ],int N, int split)
{
for(int K = 0; K<N; K++)
if(K<split)
A[K] += K;
else
A[K]*= K; }
void display(int A[ ] ,int N)
{
for(int K = 0; K<N; K++)
(K%2== 0) ?cout<<A[K]<<"%" : cout<<A[K]<<endl;
}
void main( )
{ int H[ ] = {30,40,50,20,10,5};
switchover(H,6,3);
display(H,6);            }
```

Ans : 30%41
52%60
40%25

2. Find the output of the following program :

```
#include<iostream.h>
void in(int x,int y, int &z)
{ x+=y;
 y- -;
z*=(x–y);
 }
void out(int z,int y, int &x)
{ x*=y;
y++;
z/=(x+y);
 }
void main()
{ int a=20, b=30, c=10;
        out(a,c,b);
cout<<a<<"#"<<b<<"#"<<c<<"#"<<endl;
in(b,c,a);
cout<<a<<"@"<<b<<"@"<<c<<"@"<<endl;
out(a,b,c);
cout<<a<<"$"<<b<<"$"<<c<<"$"<<endl;  }
```

3. Find the output of the following program:

```
#include <iostream.h>
struct PLAY
{ int Score, Bonus;};
void Calculate(PLAY &P, int N=10)
{
        P.Score++;P.Bonus+=N;
}
void main()
{
        PLAY PL={10,15};
        Calculate(PL,5);
        cout<<PL.Score<<":"<<PL.Bonus<<endl;
        Calculate(PL);
        cout<<PL.Score<<":"<<PL.Bonus<<endl;
        Calculate(PL,15);
        cout<<PL.Score<<":"<<PL.Bonus<<endl;
}
```

4. Find the output of the following program:

```
#include <iostream.h>
void Changethecontent(int Arr[], int Count)
{
for (int C=1;C<Count;C++)
Arr[C-1]+=Arr[C];
}
void main()
{
int A[]={3,4,5},B[]={10,20,30,40},C[]={900,1200};
Changethecontent(A,3);
Changethecontent(B,4);
Changethecontent(C,2);
for (int L=0;L<3;L++) cout<<A[L]<<'#';
        cout<<endl;
for (L=0;L<4;L++) cout<<B[L] <<'#';
        cout<<endl;
for (L=0;L<2;L++) cout<<C[L] <<'#';
                }
```

5 Find the output of the following program:

```
#include <iostream.h>
struct GAME
{ int Score, Bonus;};
void Play(GAME &g, int N=10)
{ g.Score++;g.Bonus+=N; }
 void main()
{ GAME G={110,50};
Play(G,10);
cout<<G.Score<<":"<<G.Bonus<<endl;
Play(G);
cout<<G.Score<<":"<<G.Bonus<<endl;
Play(G,15);
cout<<G.Score<<":"<<G.Bonus<<endl; }
```

Application Based Questions(2 Marks Based on random function)

1. Observe the following C++ code and find out , which out of the given options i) to iv) are the expected correct output. Also assign the maximum and minimum value that can be assigned to the variable 'Go'.

```
void main()
{ int X [4] ={100,75,10,125};
int Go = random(2)+2;
for (int i = Go; i< 4; i++)
cout<<X[i]<<"$$";
}
```

Ans :
(iv)is the correct option.
Minimum value of Go = 2
Maximum value of Go = 3

(i) 100$$75 (ii) 75$$10$$125$$ (iii) 75$$10$$ (iv)10$$125$

2. In the following program, if the value of N given by the user is 15, what maximum and minimum values the program could possibly display?

```
#include <iostream.h>
#include <stdlib.h>
void main()
{
  int N,Guessme;
  randomize();
  cin>>N;
  Guessme=random(N)+10;
  cout<<Guessme<<endl;
}
```

3. In the following program, if the value of N given by the user is 20, what maximum and minimum values the program could possibly display?

```
#include <iostream.h>
#include <stdlib.h>
void main()
{
  int N,Guessnum;
  randomize();
  cin>>N;
  Guessnum=random(N-10)+10;
  cout<<Guessnum<<endl;
}
```

4. Read the following C++ code carefully and find out, which out of the given options (i) to (iv) are the expected correct output(s) of it. Also, write the maximum and minimum value that can be assigned to the variable Taker used in the code :

```
 void main()
{     int GuessMe[4]={100,50,200,20};
int Taker=random(2)+2;
for (int Chance=0;Chance<Taker;Chance++)
cout<<GuessMe[Chance]<<"#";  }
```

(i) 100# (ii) 50#200# (iii) 100#50#200# (iv) 100#50