Q.2 a. Differentiate between: **(3)**
(i) C and C++
(ii) Insertion and Extraction operator
(iii) Polymorphism and Abstraction
(iv) Source File and Object File
(v) Bitwise and Logical operator

Answer:
(i) C and C++

C	C++
1. C is Procedural Language.	**1.** C++ is non Procedural i.e Object oriented Language.
2. No virtual Functions are present in C	**2.** The concept of virtual Functions are used in C++.
3. In C, Polymorphism is not possible.	**3.** The concept of polymorphism is used in C++. Polymorphism is the most Important Feature of OOPS.
4. Operator overloading is not possible in C.	**4.** Operator overloading is one of the greatest Feature of C++.
5. Top down approach is used in Program Design.	**5.** Bottom up approach adopted in Program Design.
6. No namespace Feature is present in C Language.	**6.** Namespace Feature is present in C++ for avoiding Name collision.
7. Multiple Declaration of global variables are allowed.	**7.** Multiple Declaration of global varioables are not allowed.
8. In C • scanf() Function used for Input. • printf() Function used for output.	**8.** In C++ • Cin>> Function used for Input. • Cout<< Function used for output.
9. Mapping between Data and Function is difficult and complicated.	**9.** Mapping between Data and Function can be used using "Objects"
10. In C, we can call main() Function through other Functions	**10.** In C++, we cannot call main() Function through other functions.

(ii). Insertion and Extraction operator

Insertion is the operation of sending characters to a stream, expressed by the overloaded insertion operator << . Thus, the following statement sends the character**'x'** to the stream **cout** :

cout << `x';

Extraction is the operation of taking characters from a stream, expressed by the overloaded extraction operator >> . The following expression (where **ch** has type**char**) obtains a single character from the stream **cin** and stores it in **ch** .

cin >> ch;

(ii). Polymorphism and Abstraction

Polymorphism means **one name many forms**.One function behaves different forms.In other words, "Many forms of a single object is called Polymorphism."

Real World Example of Polymorphism:

Example-1:

- A Teacher behaves to student.
- A Teacher behaves to his/her seniors.

Here teacher is an object but attitude is different in different situation.

Abstraction:

Abstraction is "To represent the essential feature without representing the back ground details."Abstraction lets you focus on what the object does instead of how it does it.Abstraction provides you a generalized view of your classes or object by providing relevant information.

Abstraction is the process of hiding the working style of an object, and showing the information of an object in understandable manner.

Abstraction means putting all the variables and methods in a class which are necessary.

For example: - Abstract class and abstract method.

(iv). Source File and Object File

Source file is a simple text file that you create with a text editor or IDE. An Object file is a binary file that the compiler creates that converts the source to object code and adds link symbol tables for public variables and functions that the source code contained. It also contains link tables for external variables and functions that the code uses.

You cannot open an object file and expect to see the original source code

(v).Bitwise and Logical operator

Bitwise operators (&, |, ^, ~, <<, >>)

Bitwise operators modify variables considering the bit patterns that represent the values they store.

operator	asm equivalent	description
&	AND	Bitwise AND
|	OR	Bitwise inclusive OR
^	XOR	Bitwise exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift bits left
>>	SHR	Shift bits right

Logical operators (!, &&, ||)

The operator `!` is the C++ operator for the Boolean operation NOT. It has only one operand, to its right, and inverts it, producing `false` if its operand is `true`, and `true` if its operand is `false`. Basically, it returns the opposite Boolean value of evaluating its operand. For example:

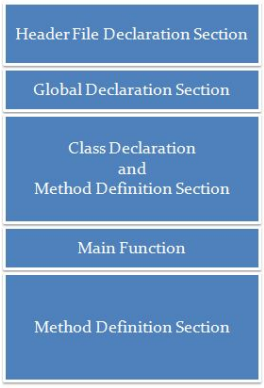
```
!(5 == 5)  // evaluates to false because the expression at its
right (5 == 5) is true
!(6 <= 4)  // evaluates to true because (6 <= 4) would be false
!true    // evaluates to false
!false   // evaluates to true
```

The logical operators && and `||` are used when evaluating two expressions to obtain a single relational result. The operator && corresponds to the Boolean logical operation AND, which yields `true` if both its operands are `true`, and`false` otherwise.
The operator `||` corresponds to the Boolean logical operation OR, which yields `true` if either of its operands is `true`, thus being false only when both operands are false. Here are the possible results of `a||b and a&&b`: are:

A	B	a && b	a || b
true	True	true	True
true	False	false	True
false	True	false	True
false	False	False	False

b.Explain the basic structure of C++ with an example.

Ans:



Example:

```
#include <iostream.h>

// main() is where program execution begins.

int main()
{
  cout << "Hello World"; // prints Hello World
  return 0;
}
```

Q.3a. With the help of example, explain for-loop.

Answer: Page 79 of text book

b.Write a program that will read the integer elements of two single-dimensional array in ascending order and merge them in the third array.

c.How a structure is different from an array?

Array	Structure
i. Data Collection	
Array is a collection of homogeneous data.	Stucture is a collection of heterogeneous data.
ii. Element Reference	
Array elements are referred by subscript.	Structure elements are referred by its unique name.
iii. Access Method	
Array elements are accessed by it's position or subscript.	Stucture elements are accessed by its object as '.' operator.

iv. Data type	
Array is a derived data type.	Structure is user defined data type.
v. Syntax	
<data_type> array_name[size];	struct struct_name { structure element 1; structure element 2; ---------- ---------- structure element n; }struct_var_nm;
vi. Example	
int rno[5];	struct item_mst { int rno; char nm[50]; }it;

Q.4a. What is the difference between passing a parameter by reference and by value?

Ans: By passing a parameter by reference, the formal argument in the function becomes alias to the actual argument in the calling function. Thus the function actually works on the original data. For example:

```
void swap(int &a, int &b) //a and b are reference variables
{
int t=a;
a=b;
b=t;
}
```

The calling function would be: **swap(m,n);**

the values of m and n integers will be swapped using aliases.

Constant Reference: A function may also return a constant reference. Example:

```
int & max(int &x,int &y)
{
If((x+y)!=0)
return x;
return y;
}
```

The above function shall return a reference to x or y.

(4)

b.What do you mean by default arguments? Illustrate with suitable examples.

Answer: Page 136, 137 of text book

c.Explain inline function and the situations where inline expansion may not work and why?

Answer: Inline functions: Whenever we write functions, there are certain costs associated with it such as jumping to the function, saving registers, pushing arguments into the stack and returning to the calling function. To remove this overhead we write an inline function. It is a sort of request to the compiler. If we declare and define a function within the class definition then it is automatically inline. We do not need to sue the term inline.
Advantage of inline functions is that they have no overhead associated with function call or return mechanism.
The disadvantage is that if the function made inline is too large and called regularly our program grows larger.
There are certain conditions in which an inline function may not work:
• If a function is returning some value and it contains a loop, a switch or a goto statement.
• If a function is not returning a value and it contains a return statement.
• If a function contains a static variable.
• If the inline function is made recursive.

Q.5 a. Define a class Rectangle which has a length and a breadth. Define the constructors and the destructor and member functions to get the length and the breadth. Write a global function which creates an instance of the class Rectangle and computes the area using the member functions.

Ans:

```
#include <iostream.h>
class CRectangle {
int width, height;
public:
CRectangle (int,int);
~CRectangle ();
friend int area(CRectangle);
};
CRectangle::CRectangle (int a, int b) {
width = a;
height = b;
}
CRectangle::~CRectangle () {
// delete width;
//delete height;
}
```

```
int area(CRectangle r)
{
return r.height * r.width;
}
int main () {
int h,w;
cout<<"enter length and breadth of rectangle";
cin>>h>>w;
CRectangle rect (h,w);
cout << "area: " <<area(rect)<< endl;
return 0;
}
```

b.Can a copy constructor accept an object of the same class as parameter, instead of reference of the object?

Ans: No. It is specified in the definition of the copy constructor itself. It should generate an error if a programmer specifies a copy constructor with a first argument that is an object and not a reference.

c.Discuss the various situations when a copy constructor is automatically invoked.

Ans A copy constructor is invoked in many situations:
• When an object is used to initialize another in a declaration statement.
• When an object is passed as a parameter to a function.
• When a temporary object is created for use as a return value of a function.
A copy constructor only applies to initialization. It does not apply to assignment.

```
Class-name(const class name ob)
{
//body of copy constructor
}
```

When an object is passed to a function a bitwise copy of that objet is made and given to the function parameter that receives the object. However there are cases in which this identical copy is not desired. For example, if the object contains a pointer to allocated memory. The copy will point to same memory as does the original object. Therefore, if the copy makes a change to the content of this memory it will be changed for the original object too. Also when the function terminates the copy will be destroyed causing its destructor to be called.

Q.6a. Define rules for operator overloading. Write a program to overload the subscript operator '[ ]'. (8)

Answer:

The rules for operator overloading are as follows:
• New operators are not created. Only the current operators are overloaded.
• The overloaded operator must have at least one operand of user defined type.
• We cannot alter the basic meaning of the operator.
• When binary operators are overloaded through a member function then the left hand operand is implicitly passed as and thus it must be an object.
• Unary operators when overloaded through member functions will not take any explicit argument.
The subscript operator can be overloaded only by means of a member function:

```
#include<iostream.h>
const int size=5;
class arrtype
{
int a[size];
public:
arrtype()
{
int i;
for(i=0;i<size;i++)
a[i]=i;
}
int operator [](int i)
{
return a[i];
}
};
int main()
{
arrtype ob;
int i;
for(i=0;i<size;i++)
cout<<ob[i]<<" ";
return 0;
}
```

b. With the help of example, explain the different types of user-defined conversions.

Answer:

Date class

```
#include<iostream.h>
#include<conio.h>
class time
{
```

```
private: int hr,min,sec;
public:
void getdata();
void operator +=(time a1);
friend ostream & operator << (ostream &out,time &t)
{
out<<t.hr<<":";
out<<t.min<<":";
out<<t.sec;
return out;
}
};
void time::getdata()
{
cout<<"ENTER THE HOURS"<<endl;
cin>>hr;
cout<<"ENTER THE MINUTES"<<endl;
cin>>min;
cout<<"ENTER THE SECONDS"<<endl;
cin>>sec;
}
void time::operator +=(time a1)
{
hr=hr+a1.min;
min=min+a1.min;
sec=sec+a1.sec;
if(sec>=60)
{
min++;
sec=sec-60;
}
if(min>=60)
{
hr++;
min-=60;
}
}
void main()
{
clrscr();
time t1,t2;
t1.getdata();
t2.getdata();
t1+=t2;
```

```
cout<<"\n RESULT OF ADDITION"<<endl;
cout<<t1;
getch();
}
```

Q.7 a. Differentiate between:

(i) Static and Dynamic

(ii) private and public inheritance

Answer: (i) Static and Dynamic binding

The information is known to the compiler at the compile time and, therefore compiler is able to select the appropriate function for a particular call at the compile time itself. This is called early binding or static binding.

The linking of function with a class much later after the compilation, this process is termed as late binding or dynamic binding because the selection of the appropriate function is done dynamically at run time. This requires the use of pointers to objects.

(ii) private and public inheritance

In private inheritence the public members of the base class become private members of the derived class. Therefore the object of derived class cannot have access to the public member function of base class.

In public inheritance the derived class inherit all the public members of the base class and retains their visibility. Thus the public member of the base class is also the public member of the derived class.

```
Class a
{
int x;
public:
void disp()
{
}
};
class y : private a
{
int c;
public:
void disp()
{
}
};
so when d.disp() // disp() is private;
```

b. What is multiple inheritance? Discuss the syntax and rules of multiple inheritance in C++. How can you pass parameters to the constructors of base classes in multiple inheritance? Explain with suitable example.

Answer: C++ provides us with a very advantageous feature of multiple inheritance in which a derived class can inherit the properties of more than one base class. The syntax for a derived class having multiple base classes is as follows:

```
Class D: public visibility base1, public visibility base2
{
 Body of D;
}
```

Visibility may be either 'public' or 'private'.
In case of multiple inheritance, the base classes are constructed in the order in which they appear in the declaration of the derived class. However the constructors for virtual base classes are invoked before any non-virtual base classes. If there are multiple virtual base classes, they are invoked in the order in which they are declared. An example program illustrates the statement.

```
#include<iostream.h>
using namespace std;
class M
{
 protected:
 int m;
 public:
 M(int x)
 {
 m=x;
 cout<< "M initialized\n";
 }
};
class N
{
 protected:
 int n;
 public:
 N(int y)
n=y;
 cout<< "N initialized\n";
 }
};
class P: public N, public M
{ int p,r;
```

```
public:
P(int a,int b,int c, int d):M(a),N(b)
{
p=c;
r=d;
cout<< "P initialized\n";
}

};
void main()
{
P p(10,20,30,40);

}
```

Output of the program will be:

```
N initialized
M initialized
P initialized
```

Q.8 a. How are template functions overloaded? Explain with a suitable example.

Answer: A template function can be overloaded using a template functions. It can also be overloaded using ordinary functions of its name. The matching of the overloaded function is done in the following manner.

a. The ordinary function that has an exact match is called.

b. A template function that could be created with an exact match is called.

c. The normal matching process for overloaded ordinary functions is followed and the one that matches is called. If no match is found, an error message is generated. utomatic conversion facility is not available for the arguments on the template functions. The following example illustrates an overloaded template function:

```
#include<iostream.h>
#include<string.h>
using namespace std;
template<class T>
void display(T x)
{
 cout<<"Template display: "<<x<<"\n";
}
void display(int x)
{
 cout<<"Explicit display: "<<x<<"\n";
}
int main()
```

```
{
 display(20);
 display(3.6);
 display('A');
 return 0;
}
```

b.What are the rules used for namespace?

Answer: A **namespace** is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries. Using namespace, you can define the context in which names are defined. In essence, a namespace defines a scope.

Defining a Namespace:

A namespace definition begins with the keyword **namespace** followed by the namespace name as follows:

```
namespace namespace_name {
   // code declarations
}
```

To call the namespace-enabled version of either function or variable, prepend the namespace name as follows:

```
name::code;  // code could be variable or function.
```

c.What are the various ways of handling exceptions? Which one is the best? Explain.

Answer: Exception handling is a mechanism that finds errors at run time. It detects an error and reports so that an appropriate action can be taken. It adopts the following ways:

a. Hit the exception
b. Throw the exception
c. Catch the exception
d. Handle the exception

The three blocks that is used in exception handling are try throw and catch. A further advancement that is made is of multiple catch handlers.

A program may have more than one condition to throw an exception. For this we can use more than one catch block in a program. An example is shown below illustrate the concept:

```
#include<iostream.h>
using namespace std;
void test(int x){
```

```
try {
if(x==1) throw x;
else
if(x==0) throw 'x';
else
if(x==-1) throw 1.0;
cout<<"End of try-block\n";
}
catch(char c) {
cout<<"Caught a character \n";
}
catch(int m) {
cout<<"Caught an integer\n";
}
catch(double d) {
cout<<"Caught a double\n";
}
cout<<""End of try-catch system\n\n;}
int main(){
cout<<"Testing multiple catches\n";
cout<<"x==1\n";
test(1);
cout<<"x==0\n";
test(0);
cout<<"x==-1\n";
test(-1);
cout<<"x==2\n";
test(2); return 0;}
```

Q.9 a. Describe the different modes in which files can be opened in C++.

Answer: Different modes in which files can be opened in C++.

ios:in	Open file for reading
ios:out	Open file for writing
ios:ate	Initial position: end of file
ios:app	Every output is appended at the end of file
ios:trunc	If the file already existed it is erased
ios:binary	Binary mode

b. Define a class Car which has model and cost as data members. Write

functions

(i) to read the model and cost of a car from the keyboard and store it a file CARS.

(ii) to read from the file CARS and display it on the screen. (8)

Answer:

```
#include<iostream.h>
#include<fstream.h>
class car
{
char model[10];
float cost;
public:
void read()
{
cout<<"\n Enter model of car";
cin>>model;
cout<<"\n Enter cost of car";
cin>>cost;
ofstream of("car.dat",ios::app);
of<<model;
of<<cost;
of.close(); }
void read_file()
{
ifstream ifs("car.dat");
while(ifs)
{ ifs>>model>>cost;
}
ifs.close();
cout<<"Model"<<model<<endl;
cout<<"Cost"<<cost;
}};
void main()
{ car c;
c.read();
c.read_file();}
```

c. What is the output of the following program segment?

```
Float pi = 3.14167234; int i =1, j=2;
cout.fill('$'); cout.ios::precision(5); cout.ios::width(10);
        cout<<i*j*pi<<'\n';
```

Answer: The output is $$$$6.2833.

TEXTBOOK

1. **C++ and Object-Oriented Programming Paradigm, Debasish Jana, Second Edition, PHI, 2005**