C++

Q:1 (A)

A-1 - Bjarne Stroustrup

A-2 - >> (extraction)

A-3 - Encapsulation mean to bound data members and functions that manipulate the data into a single unit.

A-4 - Op :- error message :- "Can not convert int to char.

Q:1 (B)

→ Abstraction means summary.

→ Data abstraction means to define data members and functions associated with them without any background detail.

Q: 1. (c)

Use :- When we want to check mutiple condition one by one at that time instead of else if ladder we may use switch statement.

Syntax :-

```
switch (expr)
{   case val1 :
        stat1 ;
        break ;
    ...
    default :
        stat default ;
        break ;
}
stat x ;
```

First it will check expr value with val1.
If it's matches then execute stat1.
If not it continues to check with every case until perfect match found.
If no case value matches than it execute default statement.

For example :-

```
char c;
cout << "enter colour code";
cin >> c;
switch (c)
{   case 'R':
        cout << "Red";
        break;
    case 'G':
        cout << "Green"; break;
    case 'B':
        cout << "blue";
        break;
    default :
        cout << "invalide code";
}
```

Page
Date

Q: 1 (D)

=> C++ defines two unary operators new and delete that perform the task of allocating and freeing the memory in a better and easier way.

* New operator :

-> An object can be created using new.
-> A data object created inside a block with new, will remain in existence until it is explicitly destroyed by using delete.

Syntax :-

```
pointervar = new datatype;
```

-> Here pointervar is a pointer variable of type datatype.

-> The datatype is any valid datatype.

-> new can be used to creat a memory space for any data type including user defined datatypes such as array, structures and classes.

Syntax :-

```
pointervar = new datatype [size];
```

* delete operator :-

-> When data is no longer needed, it is destroyed to release the memory space for reuse.

Syntax :-

```
delete pointervar ;
```

-> If we want to free a dynamically allocated array, we must use the following form of delete

```
delete [size] pointervar ;
```

-> Here the size specifies the number of elements in the array to be freed.

ex

```
# include <iostream.h>
void main()
{   int *a = new int ;
    *a = 100 ;
    cout << " a = " << *a ;
    delete *a ;
}
```

Q: 2 (A)

A-1 :- Cin is inbuilt object of istream class, treats data as a stream of characters.

A-2 :- Inline function is used to reduce the function call overhead.

A-3 :- 3

A-4 :- A variable can be declared as a reference by puting &.

example :-

data type & varname = original var

ex float &sum = total;

Q: 2 (B)

	POP	OOP
Name	Procedure oriented programming.	Object oriented programming.
accomplish	It is accomplished with functions.	Programs are divided into objects.
approach	Follows top-down approach	Follows bottom-up approach

-> Data move openly around from one function to another	Data is hidden and cannot accessed by external functions.

Q: 2 (D)

-> C++ data types can be divided into following categories :-

1) Basic datatypes
2) Userdefine datatypes
3) Derived datatypes.

1) Basic datatypes :-

-> Basic datatypes of C++ can be divided into main three types :-

i) integers :- This type is useful to store whole numbers.

ii) Real :- This type is useful to store real numbers.

iii) characters :- This type is useful to store alphabets, digits and symbols.

-> All these are further divided into sub types. Following table describe actual data type, memory occupied, range of the type.

	Type	size (In Bytes)	Range
1	unsigned short int	2 bytes	0 to 65,535
2	short int	2 bytes	-32,768 to 32767
3	unsigned long int	4 bytes	0 to 4,294,967,295
4	long int	4 bytes	-2,147,483,648 to 2,147,483,647
5	int (16 bit)	2 bytes	-32,768 to 32,767
6	int (32 bit	4 bytes	-2,147,483,648 to 2,147,283,647
7	unsigned int (16 bit)	2 bytes	0 to 65,535
8	unsigned int (32 bit)	4 bytes	0 to 4,294,967,295
9	signed char	1 byte	-128 to 127
10	unsigned char	1 byte	0 to 255
11	Float	4 bytes	3.4 e-38 to 3.4e38
12	Double	8 bytes	1.7e-308 to 1.7e308
13	long double	10 bytes	3.4 e-4932 to 1.1 e4932.

2) User defined datatypes :-

-> Class, struct and union are user defined data types.

3) Derived datatypes :-

-> Array and pointer are known as derived data types.

Q: 2 (D)

=> Overloading refers to the use of the same thing for different purposes.

=> C++ also permits overloading of functions.

=> This means that we can use the same function name to create function that perform a variety of different tasks.

=> This is known as function overloading.

=) Using the concept of overloading, we can design a family of functions with one function name but with different argument list.

=) The function would perform different operations depending upon the argument list in the function call.

=) A function call first matches the prototype having the same number and type of arguments and then calls the appropriate function for execution.

example

```
# include <iostream.h>
# include <conio.h>
int add (int a, int b);
int add (int a, int b, int c);
void main()
{
    int a,b,c,d,e;
    clrscr()
    cout<<"enter three numbers";
    cin>>a>>b>>c;
    d= add (a,b);
    e= add (a,b,c);
    cout<<"two no sum =" <<d;
    cout<<"three no sum =" <<e;
    getch();
}
```

```
int add (int a, int b)
{
    return (a+b);
}

int add (int a, int b, int c)
{
    return (a+b+c);
}
```

Q: 3 (A)

A-1 :- A class is a way to bind data and its associated functions together.

A-2 :- private

A-3 :- an object of its associated class is created.

A-4 :- . (Dot)

Page
Date

Q: 3 (B)

=) The memory for objects is allocated when they are declared and not when the class is specified.

=) This statement is partly true.

=) Actually the member functions are created and placed in the memory space only once when they are defined as part of a class specification.

=) Only space for member variables is allocated separately for each object.

Q: 3 (C)

=) A destructor as the name suggest, is used to destroy the objects that have been created by constructor.

=) Like, a constructor, the destructor is a member function whose name is the same as the class name but is preceed by a tilde(~).

A destructor never takes any argument nor does it return any value.

It will be involed implicitly by the compiler upon exit from the program to clean up storage that is no longer accecible.

example

```
#include<iostream.h>
#include<conio.h>
class A
{   public:
     A()
     { cout<<"\n obj created";
     }
     ~A()
     { cout<<"\n obj destroyed";
     }
};
void main()
{     clrscr();
      A a1;
      A a2;
      {
          A a3;
      }
      A a4;
      {
          A a5;
      }
      A a6;
      getch();
}
```

Q: 3 (D)

=> The private member cannot accessed from outside class. That is a non member function can't have an access to the private data of class.

=> To make an out side function 'friendly' to a class we have to simply declare this function as a friend of the class.

=> A friend function of a class is defined outside the class scope but it has the right to access all private and protected members of the class.

=> Even though the prototype for friend function appears in the class definition, friends are not member functions

=> To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend.

=> The function is defined elsewhere in the program like a normal C++ function.

=> The function definition does not use either the keyword friend or the scope operator.

example

```
#include <iostream.h>
#include <conio.h>
class A
{ int a;
  public:
      void get()
      { cout<<"enter a" ;
        cin>>a ; }
      void put () {
       cout<<" a = "<<a ; }
      friend A sum (A x , A y );
};
A sum (A x, A y) {
 A z;
 z.a = x.a + y.a
 return z; }
void main() {
  A x, y, z;
 clrscr();
  x.get();
  y.get();
  z= sum (x,y)
cout<<"x.a = "<< x.put()<<"y.a = "<<y.put ;
 cout<<"obj z " ;
  z.put();
  getch();
   return 0;
}
```

Page
Date

Q: 4 (A)

A-1 :- Friend

A-2 :- Friend

A-3 :- arrow

A-4 :- O

Q: 4 (B)

=) Class object can be initialized too.

=) That is, the initial value of an object may be provided during runtime.

=) One advantage of dynamic initialization is that we can provide various initialization formats, using overloaded constructors.

Q:4 ( )

=) By declaring a function member a static, you make it independent of any particular object of the class.

=) A static member function can be called even if no object of the class exist and the static functions are accessed using only the class name and the scope resolution operator.

=) A static member function can only access static data member, other other static member function and any other function from outside of the class.

example

```
# include <iostream.h>
# include <conio.h>
class A
{  static int a;
   int b;
   public :
      void disp ():
      {  cout<<"\n a=" <<a++;
         cout<<"\n b=" <<b++; }
      static void show ()
      {  cout <<"\n a=" <<a++;
         cout<<"\n b=" <<b++; }
};
```

```
int A :: a;
int main()
{
    clrscr();
    A x, y, z;
    x.disp();
    y.disp();
    z.disp();
    A :: show();
    x.show();
    getch();
    return 0;
}
```

Q: 4 (D)

Constructor :-

=> A constructor is a special member function whose task is to initialize the object of its class.

=> It is special because its name is the same as the class name.

=> The constructor is invoked whenever an object of its associated class is created.

=> It is called constructor because it constructs the value of data members of the class.

ex
```
class A
{   int m,n;
    public:
        A()
        {  m=0;
           n=0; }
        void disp()
        {
            cout<<"m="<<m<<"n="<<n; }
};
void main()
{   A x;
    x.disp(); }
```

=> There are 5 types of constructor.

1) Default constructor :-

=> A constructor without any argument is called default argument.

ex
```
class A
{ public:
      A()
      {     }  };
```

2) Parameterized constructor :-

=> The constructor that can take arguments are called parameterized constructors.

ex
```
class A
{ int a;
  public :
    A (int x)
    {
        a = x; }
};
```

3) Multiple constructor :-

```
class A
{ int m;
  public :
    A ()
    { m = 0; }
    A (int x)
    { m = x; }
};
```

=> When there are two constructor functions in same class it is called multiple constructor

4) Dynamic Constructor :-

=> The constructor which allocates memory at run time with the help of new operator is called dynamic constructor.

ex

```
class S
{ char * name;
  int len;
  public
      string (char *s)
    {  len = strlen(s);
       name = new char [len+1];
       strcpy( name,s);
    }
```

5) Copy constructor :-

=> A copy constructor is used to declare and initialize an object from another object.

=> A copy constructor takes a reference to an objectof the same class as itself as an argument.

eg
```
class A { int a, b;
  public :
      A ( A&x ) { a=x.a ; b=x.b ; } };
```

Q:5 (A)

A-1 :- Operator function is used to overload an operators.

A-2 :- Overloaded casting operator function / conversion function.

A-3 :- Hierarchical inheritance.

A-4 :- private.

Q:5 (B)

1) . (membership operators)
2) .* (pointer to member access operators)
3) :: (scope resolution operators)
4) ?: (condition operators).

Q: 5 (c)

=> Only existing operators can be overloaded. New operators cannot be created.

=> The overloaded operators must have at least one operant that is of user defined type.

=> We cannot change the basic meaning of an operator.

=> Overloaded operators follow the syntax rules of the original operators.

=> '., .*, ::, ?:' operators cannot be overloaded.

=> '=, (), [], ->' are overloaded through member function.

=> Unary operators, overloaded through member function take no explicit argument, but those overloaded through friend function take one argument.

=> Binary operators, overloaded through member function take one argument, but those overloaded through friend function take two arguments.

Date

Q:5 (D)

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

class String
{    int len;
     char *s;
     public:
          String()
          { len = 0;
            s = new char [len+1]; }
          String (char * x)
          { len = strlen(x);
            s = new char [len+1];
            strcpy (s,x); }
          void disp ()
          { cout << "String = " << s; }
          String operator + (String x);
          friend int operator == (String x, String
};
String String :: operator + (String x)
{    String z;
     z.len = len + x.len;
     z.s = new char [len+1];
     strcpy (z.s, s);
     strcat (z.s, x.s);
     return z;
}
```

```
int operator == (string x, string y)
{   if (strcmp(x.s, y.s) == 0)
        return 1;
    else
        return 0;
}
void main()
{   char *x, *y;
    clrscr();
    cout<<"enter two strings";
    cin>>x>>y;
    string a1(x), a2(y), a3;
    a3 = a1 + a2;
    cout<<"obj a1 = "; a1.disp();
    cout<<"obj a2 = "; a2.disp();
    cout<<"obj a3 = "; a3.disp();
    if (a1 == a2)
        cout<<"equal";
    else
        cout<<"not equal";
    getch();
}
```

Q: 6 (A)

A-1 :- A class who is derived class and also serves as a base class is called intermediate base class.

A-2 :- Multiple initialization list

A-3 :- Protected visibility mode is used to make private member inheritable.

A-4 Private.

Q: 6 (B)

=) An object can be a collection of many other object.

=) This is a class can contain objects of other classes as its members.

=) All object of a class can contain other object.

=) This kind of relationship is called containership.

Q: 6 (c)

=> The conversion from basic type to class type is accomplish with the help of constructor function.

For example :-

```
A (int x)
{
    a = x
}
```

The state ment A x = 10 will call above constructor function and convert int type to class A type.

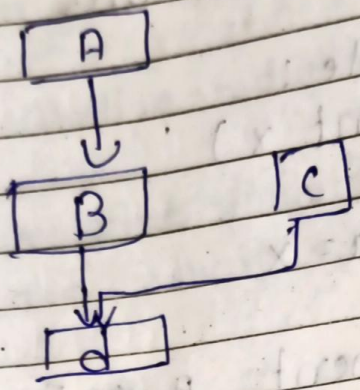
The constructor used for the type conversion take single argument whose type is to be converted.

ex

```
class A
{ int a;
  public :
      A (int x)
      { a = x ; }
  void disp ()
      { cout << "a : " << a; };
void main ()
{ A x = 10
  x. disp ( ); }
```

Q:6 (D)

Combination of two or more types of inheritance is known as hybrid inheritance.



The class A serve as a base class for the derived class B, which in turn serve as a base class for the derived class D & The class C is also serve as a base class for the derived class D.

example

```
class A
{    int a;
     public:
          void getA();
          void putA();
};
void A:: getA()
{    cout<<" enter a";
     cin>>a; }
```

```
void A::putA()
{
    cout<<"\n a= "<<a;
}

class B : public A
{  int b;
   public:
      void getB()
      { cout<<"Enter b";
        cin>>b; }
      void putB()
      { cout<<"\n b= "<<b; };

class C
{  int c;
   public:
      void getc()
      {  cout<<"enter c";
         cin>>c;  }
      void putc()
      {  cout<<"\n c= "<<c;  };

class D : public B, public C
{ int d;
  public:
      void getD()
      { cout<<"enter d";
        cin>>d; }
```

```
void putD ()
{   cout<<"\n d=" <<d;   }
};
void main ()
{   D x;
    x.getA();
    x.getB();
    x.getC();
    x.getD();
    x.putA();
    x.putB();
    x.putC();
    x.putD();
    getch();   }
```

class	privat	public
A	a	getA(), putA()
B: public A	b	getB(), putB(), getA(), putA()
C	c	getC(), putC()
D: public B, public C	d	getD(), putD(), getA(), putA(), getB(), putB(), getC(), putC().

Q: 7 (A)

A-1 Run time type information.

A-2 Istream, ostream.

A-3 Used to specify a character that is used to fill the unused portion of a field.

A-4 A stream is a sequence of bytes.

Q: 7 (B)

=) A pointer can point to an object created by a class. It is known as pointer to objects.

=) A x ; make an x object of A class.

=) To declare pointer consider following statement.

A *p ;

=) We can also use an object pointer to access the public members of an object.

A*p = &x

=) The pointer p is initialized with the address of x.

ex

```
class A
{ int a;
  public
    void getdata()
    { cout<<"enter a";
      cin>>a; }
    void putdata()
    { cout<<"\n a = "<<a; }
};
void main()
{   A x;
    A *p = &x;
    clrscr();
    p->getdata();
    p->putdata();
    getch();
}
```

Q: 7 (C)

-> We can design our own manipulators for certain special purpose.

-> The general form for creating a manipulator without any arguments is :

```
ostream & manipulator (ostream & output)
{
    ... ... (code)
    return output ;
}
```

-> Here, the manipulator is the name of the manipulator under creation.

-> The following function defines a manipulator called unit that displays "inches".

ex
```
ostream & unit (ostream & output)
{
    output << "inches";
    return output ;
}
```

Here the statement cout << 36 << unit ; will produce output 36 inches.

Q: 7 (D)

=) A virtual function is useful to achieve polymorphism.

=) When we use the same function name in both the base and derived classes, the function in base class is declared as virtual using the keyword virtual preceding its normal declaration.

=) When a function is made virtual, c++ determines which function to use at run time based on the type of object pointed to by the base pointer, rather than the type of the pointer.

=) Thus, by making the base pointer to point through different objects, we can execute different versions of the virtual function.

=) We must access virtual functions through the use of a pointer declared as pointer to the base class.

example

```
class A
{   public :
        virtual void show ().
        {
            cout << "In show class A";
        }
        void disp ()
        {
            cout << " disp A";
        }                          };

class B : public A
{   public :
            void show ()
            {  cout << "show B class";
            }
            void disp ()
            {
                cout "disp B";
            }                      };
void main ()
{   A x, * pa;
    B y, * pb;
    clrscr ();
    pa = & x;
    pa -> show ();
    pa -> disp ();
    pa = & y;
```

```
pa->show();
pa->disp();
pb = &y;
pb->show();
pb->disp();
getch();
}
```

Q: 8 (A)

A-1 :- A class containing pure virtual functions can not be used to declare any objects of its own. such classes are called abstract class.

A-2 :- Width fills rest of the characters with spaces if the output value is of less size than specified width.

A-3 :- setfill()

A-4 :- Show + sign for positive nos.

Q: 8 (13)

1) get()
2) getline()
3) iread()

Q: 8 (c)

=> When a binary operator overloaded using a member function, we pass only one argument to the function.

=> C++ uses a unique key word 'this' to represent an object that invoked a member function.

=> 'This' is a pointer that points to the object for which this function was invoked.

=> For example, the function call x.max() will set the pointer this to the address of the object x.

=> The pointer this acts as an implicit argument to all the member function.

Page
Date

example

```
class A
{   int a;
    public:
        void getdata(int a)
        { this -> a = a; }
        void putdata()
        { cout << "a = " << a; }
        A max(A y)
        { if(y.a > a)
            return y;
          else
            return *this; }
};
void main()
{ int x1, x2;
  clrscr();
  cout << "Enter two int no";
  cin >> x1 >> x2;
  A x, y, z;
  x.getdata(x1);
  y.getdata(x2);
  x.putdata();
  y.putdata();
  z = x.max(y);
  z.putdata();
  getch();
}
```

Q: 8 (D)

=> The setiosflag() method is iomanip library in c++ used to set the ios library format flags specified as the parameter to this method.

=> This method accepts Formul_flag as a parameter which is the ios library format flag to be set by this method.

=> This method does not returns anything. It only acts as stream manipulators.

example :-

```
# include <iomanip.h>
# include <ios.h>
# include <iostream.h>
int main()
{   int num = 50;
    cout << "Setting showbase flag " << setiosFlags
            (ios::showbase) << num << endl;
    return 0;
}
```

output : setting showbase flag 0x32.

Q: 9 (A)

A-1 :- Returns true when an input or output operation has failed.

A-2 :- Standard Template Library.

A-3 :- Iterators are used for working upon a sequence of values.

A-4 :- record

Q: 9 (B)

	Text File.	Binary File.
→	A text file stores data in alphabets, symbols, digits by their ASCII values.	A binary file contains a sequence of bytes which are either 0 or 1.
→	This file is in human readable format.	This file is not in human readable format.
→	A small error in text file can be recognized and eliminated when seen.	A small error in binary file corrupts the file and is not easy to detect.

Page
Date

Q: 9 ( )

=> C++ supports a feature that facilitated the supply of arguments to the main ( );

=> These arguments are supplied at the time of invoking the program.

=> The command line arguments are typed by the user and are delimited by a space.

=> The first arguments always the filename and contains the program to be executed.

=> Four command line arguments main ( ) takes two argument as shown below.

```
main (int argc, char *argv[])
```

=> The first argument argc represents the number of arguments in the command line.

=> The second argument argv is an array of char type pointers that points to the command line arguments.

Q: 9 (D)

```
# include <iostream.h>
# include <fstream.h>
# include <conio.h>
class stud
{       int rn;
        char nm[20];
        public :
                void get ()
                { cout<<" enter rlno & name ";
                  cin>>rn>>nm; }
                void put()
                { cout<<endl<<rn<<nm; }
};
void main()
{       stud s;
        char c;
        clrscr();
        ofstream f1("recwrite2.txt");
        while(1)
        { s.get();
          f1.write((char*)&s, size of(s));
          cout<<"continue";
          cin.get();
          cin>>c;
          if (c!='y')
                break; }
```

```
f1.close()
ifstream f2("readwrite2.txt");
cout << "Roll no name";
while(f2.read((char*)&s, sizeof(s)))
{
    s.put();
}
f2.close();
getch();
}
```

Q: 10 (A)

A-1 :- Opening file using constructor and opening file using open()

A-2 :- We can get file length by putting seekg at end of file and storing length with tellg().

ex
```
f2.seekg(0, ios::end);
filelength = f2.tellg();
```

A-3 :- tellg() gives the current possition of the get pointer.

A-4 :- ios::beg

Q: 10 (B)

1) Algorithms
2) Containers
3) Functions
4) Iterators.

Q:10 (c)

Page
Date

=> Containers or container classes stores objects and data. There are in total seven standard "first-class" container classes and three container adaptor classes and only seven header files that provide access to these containers or container adaptors.

=> Sequence Containers :- Implement data structures which can be accessed in a sequential manner.
- vector
- list
- deque
- arrays
- forward list

=> Container Adaptors :- Provide a different interface for sequential containers.
- queue
- priority queue
- stack

=> Associative Containers :- Implement sorted data structures that can be quickly searched.
- set
- multiset
- map
- multimap.

Q: 10 (D)

=> Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.

=> A template is a blueprint or formula for creating a generic class or a function.

=> The library containers like iterators and algorithms are example of generic programming and have been developed using template concept.

=> You can use templates to define functions as well as classes, let us see how they work.

* Function Template :-

=> The general form of a template function definition is shown here :-

```
template <class type>
ret-type func-name (parameter list)
{
   // body of function
}
```

Here type is a placeholder name for a data type used by the function. This name can be used within the function definition.

* class template :-

=> Just as we can define function template, we can also define class templates.

=> The general form of a generic class declaration is shown here.

```
template < class type >
class class-name
{ ... } ;
```

=> Here, type is the placeholder type name, which will be specified when a class is instantiated.

=> A class created from a class template is called template class.