CBCS SCHEME

USN | 2 | V | D | 2 | 3 | C | S | C | 3 | 2 |

BCS306B

Third Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025
Object Oriented Programming with C++

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	What is meant by constructor? Discuss the types of constructors in C++ with example.	07	L2	CO1
	b.	Demonstrate with example, how friend functions, friend classes and inline functions are useful in C++.	07	L2	CO1
	c.	With a sample program, demonstrate the general format of a C++ program.	06	L3	CO1
		OR			
Q.2	a.	List the various access specifiers supported by OOPs. Illustrate their use.	07	L2	CO1
	b.	What is the significance of static data and member functions in C++? Explain.	07	L2	CO1
	c.	What is Object Oriented Programming? Explain its features.	06	L2	CO1
		Module – 2			
Q.3	a.	Develop a object oriented program to find the smallest and biggest among array elements.	07	L3	CO2
	b.	Discuss pointers to object with example code. Also discuss their advantages.	07	L2	CO2
	c.	Explain different array type in C++ with suitable example code snippet.	06	L3	CO2
		OR			
Q.4	a.	What is "this" pointer? Illustrate the use of "this" pointer in C++.	07	L2	CO2
	b.	Discuss when two or more functions are said to be overloaded. Identify the causes of ambiguity in function overloading.	07	L2	CO2
	c.	What is a dynamic constructor? Explain with an example program.	06	L3	CO2
		Module – 3			
Q.5	a.	List the operators in C++ that can not be overloaded. Develop a C++ program using "Time" class, to overload the '+' and '-' operators.	07	L3	CO3
	b.	Illustrate the role of access-specifiers in different level of inheritances.	07	L2	CO3
	c.	Discuss various types of inheritances with suitable example codes.	06	L3	CO3
		OR			
Q.6	a.	Illustrate the use of constructors and destructors in inheritance in C++.	07	L2	CO3
	b.	Develop a program in C++ to derive a class "Manager" from class "Person" and "Employee". Consider suitable data members and member-functions for the classes.	07	L3	CO3
	c.	Explain "Virtual base Class" with an example.	06	L3	CO2

		Module – 4			
Q.7	a.	Design a C++ program demonstrating the use of the Pure Virtual function using base and derived classes. Also explain the code.	07	L3	CO4
	b.	What is polymorphism in C++? Explain its types with example.	07	L3	CO4
	c.	Explain virtual function in C++. Discuss what is early and late binding.	06	L2	CO4
		OR			
Q.8	a.	What are generic functions? Demonstrate the use of generic function in swapping two variables of any type.	07	L2	CO4
	b.	What is typename and export keyword? Discuss their usage. Discuss the advantages of using templates in C++.	07	L2	CO4
	c.	Explain the use of a class template. Also explain class template with suitable code snipplet.	06	L3	CO4
		Module – 5			
Q.9	a.	Explain the fundamentals of exception handling in C++. Analyze the benefits of exception handling.	07	L2	CO5
	b.	Discuss different standard exceptions in C++.	07	L2	CO5
	c.	Develop a C++ program to demonstrate the usage of try, catch and through to handle exceptions.	06	L3	CO5
		OR			
Q.10	a.	What are the different file opening modes in C++? Compare and contrast file opening modes.	07	L2	CO5
	b.	Explain file streams with example.	06	L3	CO5
	c.	Develop a C++ program to create a text file, check file is created or not, if created, write some text in to the file and the read and display the text from the file.	07	L3	CO5

* * * * *

Third Semester B.E. Degree Examination Jan - 2025
Object Oriented Programming with C++
Sub. code: BCS306B.

Q.1
a) What is meant by constructor? Discuss the types of constructors in C++ with example.

Soln: In C++, a constructor is a special member function that is automatically invoked when an object of the class is created.
The constructors are used to initialize the objects data members & allocate resources if necessary. (1)

Types of constructors:
① Default constructor is a constructor that takes no arguments. If no constructor is explicitly defined, C++ provides a default constructor.
Ex:

```
class Rectangle
{
  public:
    int len, wid;

    Rectangle ()
     {
       len=10;
       wid=5;
     }
    void display()
    {
      cout<<" length: "<< len;
      cout<<" Width: "<< wid;
    }
};
int main()
{
  Rectangle r;
  r.display();
  return 0;
}
```

(2)

② Parameterized constructor: it is a constructor that accepts parameters & allows to initialize an object with custom values at the time of creation.

Ex:

```
class Rectangle
{
  public:
    int len, wid;
    Rectangle(int l, int w)
    {
       len=l;
       wid=w;
    }
    void display()
    {
      cout<<"length: "<<len;
      cout<<"width: "<<wid;
    }
};
int main()
{
    Rectangle r(10,20);
    r.display();
    return 0;
}
```

②

① copy constructor: is used to initialize a new object as an copy of an existing object.

Ex:

```
class Rectangle
{
  public:
   int len, wid;
   Rectangle (int l, int w)
   {
      len = l;
      wid = w;
   }
```

```
Rectangle (const Rectangle &r)
{
    len = r.len;
    wid = r.wid;
    cout<< " copy constructor called "<<endl;
}
void display( )
{
   cout<< "length : "<< len;
   cout<< " width : "<<wid;
}
};
int main( )
{
   Rectangle r(30, 15);
   r.display( );

   Rectangle r2 = r;
    r2.display( );
   return 0;
}
```

(2)

1
b) Demonstrate with an example, how friend functions, friend classes & inline functions are useful in C++.

Soln * A friend function is a non-member function of the class but it is allowed to access the private & protected members of that class. (3)

* A friend class allows one class to have full access to the private & protected members of another class, making it useful for operations.

* Inline functions eliminate overhead of function calls. as they are replaced by the actual code of the function call.

Ex:

```
class Box
{
    double len;
    public:
        Box(double l) : len(l){ }
```

2.

```
friend void print( Box b)
{
    cout<< "length of the box " << b.len<<endl;
}
};
int main ()
{
   Box b( 10.5);
   print(b);
   return 0;
}
```

(c) With a sample program, demonstrate the general format of C++ program (6)

Soln: General Format of C++ program:

```
#include<iostream>
using namespace std;
class Rectangle
{
  double len;
  double wid;
  public:
    Rectangle (double l, double w)
    {
        len = l;
        wid = w;
    }
    double Result()
    {
       return len*wid;
    }
};  void Result ()
  int main()
  {
      Rectang r(5.0, 3.0);
       r. Result();
       return 0;
  }
```

(6)

List various access specifiers supported by OOP. Illustrate their use. (7)

2. Access specifies:

(1) Public Access Specifier: Members declared as public are accesible from anywhere in the program.

(2) Private: Members declared as private are not accesible from outside the class. They can only be access by members functions or friend functions. (3)

(3) Protected: Members declared as protected are accesible within the class itself & derived by classes. They are not accesible from outside the class directly

Ex:

```
class Box
{
private:
    double len;
public:
    void set(double l)
    {
        len = l;
    }
double getlen()
{
    return len;
}
};
int main()
{
    Box b;
    b.set(15.0);
    cout<<"length of box" << b.getlen();
    return 0;
}
```

(4)

(b) What is the significance of static data & member functions in C++? Explain. (2)

Soln Static data member is a class variable that is shared by all objects of the class. There is only one copy of the static data member, regardless of how many objects of the class are created.

Static member function is a function that can be called on the class itself, without the need for an object of the class. Static member functions can only access other static members of the class. (3)

ex:

```
class counter
{
    static int count=0;
    public:
        counter()
        {
            count++;
        }
        static int getcount()
        {
            return count;
        }
};
int main()
{
    counter c1,c2;
    cout<<"count value "<<counter::getcount()<<endl;
    cout<<endl;
    return 0;
}
```

(4)

P.M. (3)

2) What is Object Oriented Programming? Explain its features. (6)

Ans: Object oriented Programming is a paradigm based on the concept of classes & objects. It helps organize code by combining data & methods into objects. (1)

Features:

→ Class: A blueprint for creating objects
→ Object: An instance of a class.
→ Encapsulation: Bundles data & methods together.
→ Inheritance: Allows one class to inherit properties & methods from another class, promoting code reusability. (2)
→ Polymorphism: Enables one function or method to have different behaviours based on the context.
→ Abstraction: Hides implementation details from the user & displays only essential features, reducing complexity.

3)a) Develop a object oriented program to find smallest & biggest among array elements. (7)

Ans:

```
class Array
{
    int *a;
    int size;
    public:
     Array(int a[], int size)
     {
         this->a = a;
         this->size = size;
      }
     int smallest()
      {
        int small = a[0];
        for(int i=1; i<size; i++)
         { if (a[i] < small)
            {
                small = a[i];
            }
         }
        return small;
      }
```

```
int largest()
{
    int large = a[0];
    for (int i=1; i<size; i++)
    {
        if (a[i] > large)
        {
            large = a[i];
        }
    }
    return large;
};
int main()
{
    int a[] = {5,3,2,9,1};
    int size = sizeof(a)/sizeof(a[0]);
    Array ar(a,size);
    cout<<" smallest no:"<<ar.smallest()<<endl;
    cout<<" largest no:"<<ar.largest()<<endl;
    return 0;
}
```

b) Discuss pointers to object with an example. Also discuss their advantages. (7)

Soln: Pointers store address of an object of a class. Using pointers, it is possible to dynamically allocate memory for objects and access members of the objects using pointer.

Ex:
```
class Rectangle
{
  public:
    int l, b;
    Rectangle(int l, int b) : l(l), b(b) {}
    int area()
    {
      return l*b;
    }
};
```

```
int main()
{
    Rectangle *r = new Rectangle(10,5);
    cout<<" Area : "<< r->area();
    delete r;
    return 0;
}
```

c) Explain different array type in C++ with suitable example. (6)

Soln: i) One dimensional Initialized Arrays: An initialized is declared with specific values assigned to its elements at the time of declaration.

```
#include<iostream>
int main()
{
    int a[] = {1,2,3,4,5};
    cout<<" Initialized array elements ";
    for(int i=0; i<5; i++)
    {
        cout<<a[i]<<" ";
    }
    return 0;
}
```

* Uninitialized arrays are declared without explicitly assigning initial values to their elements.

4)(a) What is this pointer? Illustrate the use of this pointer.

Soln: When a member function is called, it is automatically passed an implicitly argument that is a pointer to the invoking object. This p is called this pointer.

Eg:
```
class Rectangle
{
    int l;
    int b;
    public:
    Rectangle(int l, int b)
    {
        this->l = l;
        this->b = b;
    }
```

```
int area()
{
    return this->l * this->b;
}
void display()
{
    cout<<"Length:"<<this->l;
    cout<<"Breadth:"<<this->b;
}
};
int main()
{
    Rectangle r(10,5);
    r.display();
    cout<<"Area:"<<r.area();
    return 0;
}
```

b) Discuss when two or more functions are said to be overloaded. Identify the causes of ambiguity in function overloading.

Soln: Function overloading occurs when two or more functions in the same scope have the same name but differ in their parameter list.

Ex:
```
class Calculator
    {
    public:
        int add(int a, int b)
        {
            return a+b;
        }
        float add(float a, float b)
        {
            return a+b;
        }
};
int main()
{
    Calculator c;
    cout<<"sum of integers"<<c.add(5,3);
    cout<<"sum of float"<<c.add(5.5f, 3.3f);
    return 0;
}
```

4) What is dynamic constructor? Explain with an example. (6)

Soln: Dynamic constructor uses dynamic memory allocation to allocate memory to objects at runtime. In C++, dynamic memory allocation is typically done using the new keyword.

eg:
```
class Array
{
  int *a;
  int size;
  public:
  Array(int s)
  {
    size=s;
    a = new int [size];
  }
  void input()
  {
    for(int i=0; i<size; i++)
    {
      cin>>a[i];
    }
  }
  void display()
  {
    for(int i=0; i<size; i++)
    {
      cout<<a[i]<<" ";
    }
  }
};
int main()
{
  int n;
  cout<<"Enter array size";
  cin>>n;
  Array obj(n);
  obj.input();
  cout<<Array elements"<<endl;
  obj.display();
  return 0;
}
```

Q.5)
a) List the operators in C++ that can not be overloaded. Develop a C++ program using Time class, to overload + & - operators.

Soln: Operators that cannot be overloaded in C++

1. Scope Resolution operator (::)
2. Member Access operator (.)
3. Pointer to Member operator (.*, ->*)
4. conditional operator (?:)
5. Sizeof operator
6. New & delete operators.

```
class time
{
    int hr;
    int min;
    public:
      time(int h, int m)
      {
        hr = h;
        min = m;
      }
     time operator + (const time & t)
     {
       int total = (hr + t.hr);
       int min = min + t.min;
     } return tot + min;

     void display()
      {
        cout << hr << " hr and " << min << " Minutes" << endl;
      }
};
int main()
{
    time t1(2, 45);
    time t2(4, 30);
    time t3 = t1 + t2;
    time t4 = t1 - t2;
    cout << " Time " << t3.display();
    cout << " Time " << t4.display();
```

b) Illustrate the role of access specifier in different level of inheritances.

Soln: In C++, the access specifiers (public, protected, private) in inheritance control how the members of a base class are inherited by a derived class.

Inheritance type	Base class Member	Derived class Access
1) Public Inheritance	public	Accessible as public
	protected	Accessible as protected
	private	Not accessible.
2) Protected Inheritance	public	Accessible as protected
	protected	Accessible as Protected
	Private	Not accessible
3) Private Inheritance	public	Accessible as private
	protected	Accessible as private
	private	Not Accessible.

c) Discuss various types of Inheritances with example.

Soln: (a) Single Inheritance, a class derives from only one base class.

(b) Multiple Inheritance, a class can inherit from more than one base class.

(c) Multilevel Inheritance, a class derives from another derived class.

(d) Hierarchical Inheritance: multiple derived classes inherit from single base class.

(e) Hybrid Inheritance: a class inherits from more than one base, & at least one of those base classes is derived from another class.

Q.6)

a) Illustrate the use of constructors & destructors in C++

Soln: A constructor is a special function called when an object of a class is created.

A destructor is a special function called when an object is destroyed.

Ex:

```
class Example
{
  public:
    Example ()
    {
      cout<<" This is constructor "<<endl;
    }
    ~Example()
    {
      cout<<" This is destructor "<<endl;
    }
};
int main()
{
  Example e;
  return 0;
}
```

```
void showManager()
{
    show();
    showSalary();
    cout<< "Manager details";
}
};
int main()
{
    Manager m("Ajay", 45, 101, 75000);
    m.showManager();
    return 0;
}
```

c) Explain virtual base class with an example.

Sol: In C++, a virtual base class is used in multiple inheritance to avoid the diamond problem. The virtual base class ensures that only one instance of the common base class is inherited.

Ex:
```
class A
{
  public:
    A()
    { cout<< "This is class A constructor";
    }
};
class B: public virtual public A
{
  public:
      B()
      { cout<< "This is class B constructor";
      }
};
class C: virtual public A
{
    C()
    {
      cout<< "This is class C constructor";
    }
};
```

b) Develop a C++ program to derive a class "Manager" from class "Person" & "Employee". consider suitable data members & member function for the classes.

Soln:

```
class Person
{
  public:
  string name;
  int age;
  Person (string n, int a) : name(n), age(a) { }
  void show()
  {
    cout<< name << "  " << age;
  }
};
class Employee
{
  public:
  int id;
  double salary;
  Employee (int id, double sal)
  {
     id = id;
     salary = sal;
  }
  void showm()
  {
    cout << id << " " << salary;
  }
};
class Manager : public Person, public Employee
{
  public:
    Manager (string n, int a, int id, double sal)
    {
      id=id;
      n= name;
      age = as
      salary = sal;
    }
```

b) What is polymorphism? Explain its type with example.

Soln: Polymorphism allows objects of different types to be treated as objects of a common base type.

There are two main types of polymorphism

① Compile time Polymorphism: occurs when the function or method to be invoked is determined at compile time.

② Run time Polymorphism: occurs when the method to be invoked is determined at runtime.

ex:
```
class Point
{
  public:
    void printer(int i)
    {
      cout<<" Print integer"<<i;
    }
    void printer(float i)
    {
      cout<<" Print float"<<i;
    }
};
int main()
{
  Point p;
  p.print(10);
  p.print(10.0);
  return 0;
}
```

c) Explain virtual function. Discuss what is early & late binding.

Soln: Virtual function is a member function in a base class that override in a derived classes. It is declared using the virtual keyword in the base class. The base class pointer calls the appropriate function in the derived class.

```
int main()
{
    C obj;
    return 0;
}
```

Q7

a) Design a C++ program demonstrating the use of the Pure Virtual function using base & derived classes.

Soln:

```
class shape
{
  public:
    virtual void draw() = 0;
};

class circle : public shape
{
  public:
    void draw()
    {
      cout << "draw circle";
    }
};

class Rectangle : public shape
{
  public:
    void draw()
    {
      cout << "draw rectangle";
    }
};

int main()
{
  circle c;
  Rectangle r;
  c.draw();
  r.draw();
  return 0;
}
```

b). What typename & export keyword. Discuss their usage. Discuss their advantages.

Soln: * typename keyword is used to specify that a name represents a type in the content of templates.

* typename is used to declare a type that will be specified later when the template is instantiated.

```
template <typename T>
T add (T a, T b)
 {
    return a+b;
 }
int main()
{
   cout<< add(3,4) << endl;
   cout<< add (3.5, 4.5) <<endl;
   return 0;
}
```

* export keyword allow template definations to be seperated from their declarations.

Ex:

```
template <typename T> export T add(T a, T b);

template <typename T> T add (T a, T b)
 {
    return a+b;
 }
```

c) Explain the use of class template. Also explain class template with code.

Soln: class template is a blueprint for creating classes that can operate on any data type.

Ex:

```
template <typename T>
class A
 {
    T value;
    public:
    A (T val)
     {
     } value = val;
```

* Early binding also called static binding occurs when the function to be called is determined at compile time.
* Here the function is not virtual.
* Function calls are resolved at the compile time.
* Late binding also called as dynamic binding occurs when the function to be called is determined at runtime.
* The function is virtual.
* Function calls are resolved at execution time based upon the actual object type.

Q.8

a) What are generic functions? Demonstrate the use of generic function in swapping two variable of any types.

Soln: Generic function can operate on different data types without re-writing for each type. It is achieved using templates.

Ex:

```
template <typename T>
void swap (T& a, T& b)
 {
     T temp = a;
     a = b;
     b = temp;
 }
int main()
{
  int x=5, y=10;
  swap (x, y);
  cout<< "x = "<< x <<"y" << y;
  double x = 5.5, y = 10.5;
  swap(x, y);
  cout<< "x =" << x << "y"<< y;
  return 0;
}
```

```
    void display()
    {
        cout<< "value" << value;
    }
};
```

9

a) Explain the fundamentals of Exception handling in C++
Analyze the benefits of exception handling.

Ans: Exception handling in C++ is a mechanism that handles runtime errors, allowing a program to continue execute or terminate.

* Exception handling includes,

→ try block: contains a code that may cause an exception.

→ throw the occurred exception thrown using keyword throw.

→ catch block: follow the try block and contains code to handle the exception. if it occurs.

Advantages:

→ Seperation of error handling code.
→ Controlled error handling.
→ Better resource management.
→ Cleaner code
→ Improved debugging.

b) Discuss different standard exceptions in C++.

Ans: ① std::exception: The root of all standard exception. Provides virtual function what() to retrieve an error message.

② std::logic_error: represents errors that can be theoretically be detected by reading the code, such as violation of logical conditions.

⑤ std:: runtime_error : represents errors that cannot be detected at compile time and occur during program execution.

c) Develop C++ program to demonstrate the usage of try, catch & throw to handle exception.

Soln:

```
#include <iostream>
int main()
{
    int n1, n2;
    cout<< "Enter two no" <<endl;
    cin>> n1 >>n2;
    try {
        if(n2==0){
            throw "Divide by zero";
        }
        int result = n1 / n2;
        cout<< "Result = " << result;
    }
    catch( const char* msg)
    {
        cout<< msg <<endl;
    }
    return 0;
}
```

Q.10 a) What are the different file opening modes in C++? Compare & contrast file opening modes.

Soln: ① ios::in (input mode): used to open a file in read mode. To open file read mode, file must exist

② ios::out (output mode): used to open a file for writing. If file exist, the contents are overwritten.

③ ios::app (Append mode): used to open a file for writing at the end of the file. If the file exist the contents will not be overwritten. The file created if it doesn't existed.

(4) ios::ate (At end mode): used to open the file for both read & write operation., the file pointer is positioned at the end of the file.

(5) ios::binary (Binary mode): Opens a file in binary mode, used for reading or writing binary files.

b) Explain file streams with example.

Ans: (1) ifstream: Input file stream used to read from file.

Syntax: ifstream file_name;

(2) ofstream: used for writing to files

Syntax: ofstream file_name;

(3) fstream: used for both reading & writing to a file.

Syntax: fstream file_name.

Ex:
```
#include<fstream>
int main()
{
  ofstream outFile("text.txt");
  if(outFile)
    {
      outFile<<"this is test"<<endl;
      outFile.close();
    }
  ifstream inFile("text.txt");
  string line;
  if(inFile)
   {
     while(getline(inFile, line))
       {
         cout<<line<<endl;
       }
     inFile.close();
   }
  return 0;
}
```

Write a C++ program to create a text file, check file is created or not, if created write some text into the file & read & display the text from the file.

Soln:

```
#include <fstream>
int main()
{
    ofstream outFile("example.txt");
    if(!outFile){
        cout<<" Error: File does not exist";
        return 1;
    }
    outFile<<" This is Test File"<<endl;
    outFile<<"writing text into file"<<endl;
    outFile.close();
    ifstream inFile("example.txt");
    if(!inFile){
        cout<<" File could not be opened";
        return 1;
    }
    string line;
    while(getline(inFile, line)){
        cout<<line<<endl;
    }
    inFile.close();
    return 0;
}
```

HOD
Computer Science & Engineering
KLS Vishwanathrao Deshpande
Institute of Technology, Haliyal