B. E.

Fourth Semester Examination, May-2007

OBJECT ORIENTED PROGRAMMING USING C++

Note : Attempt any five questions.

Q. 1. What do you understand by a name space? Discuss the syntax for defining a namespace. Discuss its use using suitable example.

Ans. Name Space :

It is a new concept introduced in ANSI C++ standards committee. This defines the scope for identifiers that are used in a program. For using the identifiers defined in name space scope we must include using directive like

```
using namespace std;
```

Here, std, is the name space where ANSIC++ standard library are defined. All ANSI C++ programs must include this directive. This will bring all the identifiers defined in std to the current global scope.

The syntax for defining namespace in general form is

```
name space namespace-name {
    //    declarations of
    //    variables, functions, classes etc.}.
```

Eg. :

```
    namespace testspace
{
    int m;
    void display (int n)
{
    cout<<n;}
}.
```

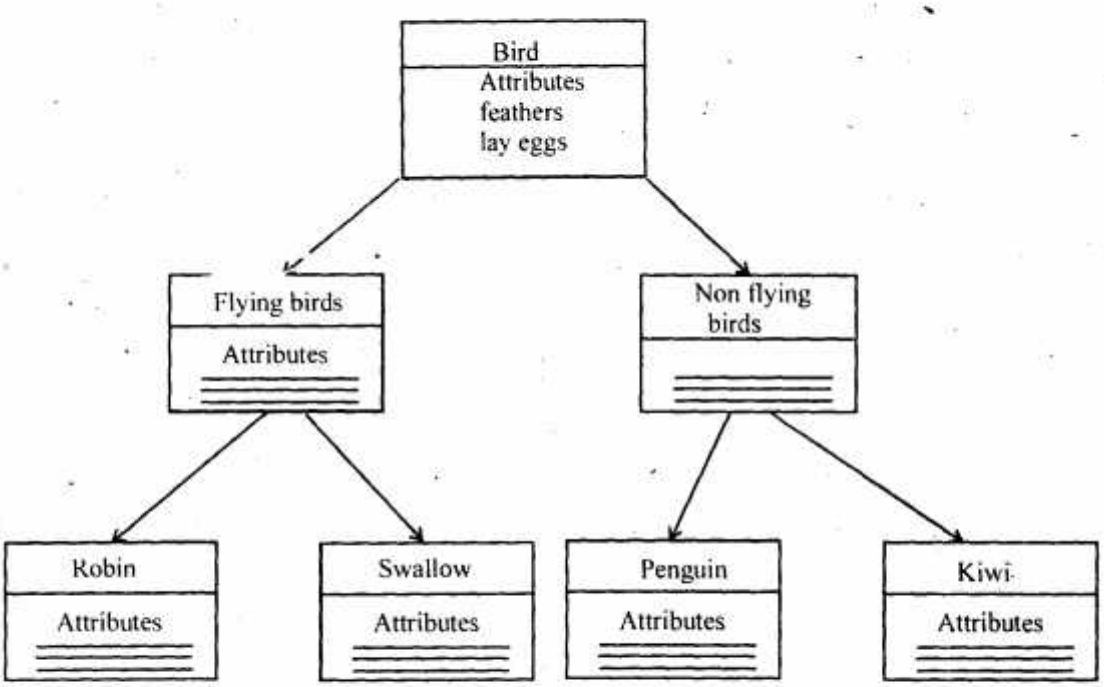
Q. 2. What do you understand by reusability? How does it help in reducing the cost of software and improving its quality? Discuss the features available in C++ to support reusability.

Ans. The concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by driving a new class from the existing one. The new class will have the combined features of both the classes. The real appeal and power of inheritance

mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants and to tailor the class in such a way that it does not introduce any side effects into rest of the classes.

Eg. : The diagram shows

Each subclass defines only those features which are unique to it. Without the use of classification, each class would have to explicitly introduce all of its features.



It is always nice if we could reuse something that already exists rather than trying to create same all over again. It would not only save time and money but also reduce frustration and increase reliability. e.g., the reuse of the class that has already been tested, debugged and used many times can save us the efforts of developing and testing the same again.

Q. 3. (a) What do you understand by a static member function of a class? Discuss their characteristics. Give an example where you can justify the use of static member functions.

Ans. Static Member Functions : A member function that is declared static has following properties :

- A static function can have access to only other static members declared in the same class.

– A static member function can be called using the class name as follows :

Class-name : : function_name.

Example :

```
# include<iostream.h>
class test
{ int code;
static int count;
public;
void set code (void)
    {    code = ++count;]
         void showcode (void)
    {    count<< 'object number:' <<code;}
         static void show count (void)
    {    Cout<<"count:" <<count;};
         int test : : count;
         int main ( ){
              test t1, t2;
              t1.set code ( );
              t2. set code ( );
              test : : show count ( );
              test t3;
              t3.set code ( ).
              test : : show count ( );
              t1.show code ( );
              t2.show code ( );
```

/

```
t3. show code ( );
return 0;
}
```

Q. 3. (b) What is a friend function? Discuss its characteristics.

Ans. Friend function :

Private members can't be accessed from the outside the class. That is, non member functions can't have an access to private data of a class. In that situation we need friend function which can access the private data members and member functions without being part of that class.

A friend function possesses following properties :

- It is not in the scope of the class to which it has been declared as friend.
- It has object as arguments.
- It can be declared either in the public or the private part of class without affecting its meaning.
- It can be invoked like a normal function without help of any object.

Q. 4. (a) What is operator overloading? Differentiate between overloading of binary operator using friend function and without using friend function.

Ans. Operator Overloading :

To define additional task to an operator, we must specify it means in relations to the class to which the operator is applied. This can be done with special function called operator function which describes the task.

The general form of an operator function is,

```
return type class name : : operator op (arglist)
{
    // function body
}
```

Friend function may be used in the place of member function for overloading binary operators, the only difference being that a friend function requires two arguments to be explicitly passed to it while a member function requires only one.

In most cases, we will get same result by use of either friend function or a member function. These are some situations where we would like to prefer friend function rather than member function. For instance, consider a situation where we need to use two different types of operands for a binary operator say one an

object and another a built in type as shown below :

A = B +2;

or

(A = B*2).

Q. 4. (b) Write a program to overload "+" operator using friend function to concatenate two strings.

Ans. Operator overloading using friend function :

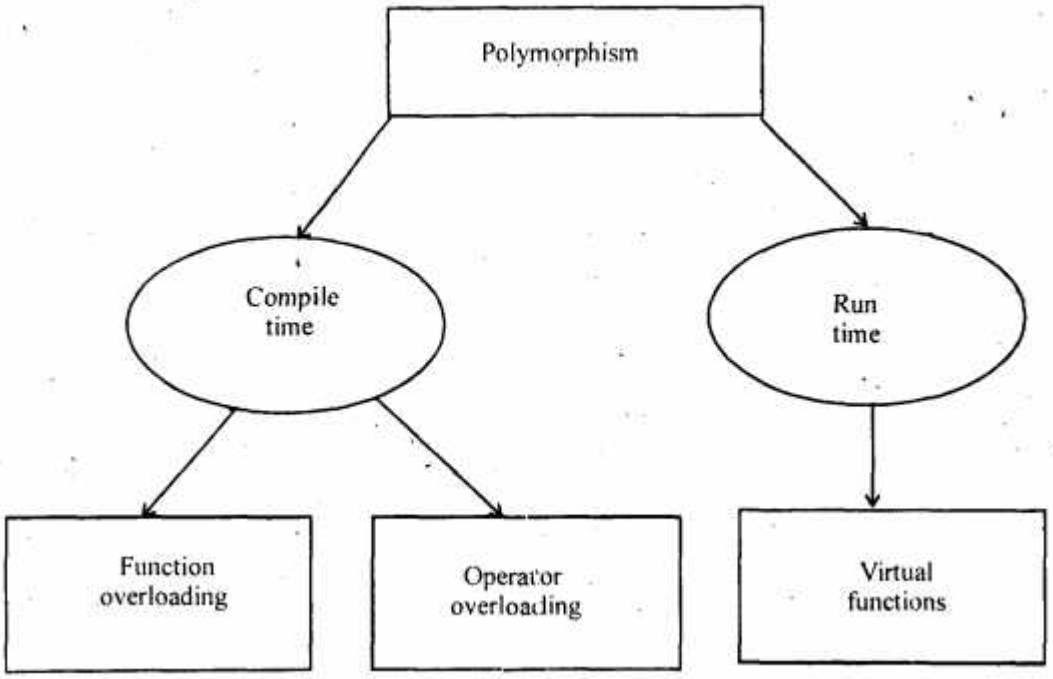
```
Class complex
{
float x, y
public :
        complex ( ) { }
        complex (float real, float imag)
        { x = real; y = image;}
        friend complex operator + (complex a complex b);
        void display (void);
        };
                complex complex : : operator + (complex a, complex b){
                return complex ((a.x+b.x), (a.y+b.y));
        }
        void complex :: display (void) {
        cout <<x<< "+j"<<y;}
        int main ( )
        {
                complex C1, C2, C3;
                C1 = complex (2.5, 3.5)
```

```
C2 = Complex (1.6 + 2.7);
C3 = Operator + (C1, C2);
Cout << C1.display ();
Cout << C2.display ();
Cout<<C3.display ();
return 0;
}
```

Q. 5. What is polymorphism? What is the advantage of using it? Differentiate between compile time and runtime polymorphism using examples.

Ans. Polymorphism :

Its a crucial features of OOPs. It means 'one name' rnultiple forms. The overloaded member functions are selected for invoking by matching arguments; both type and number. This information is known to the compiler at compile time and therefore computer is able to select the appropriate function for a particular call at the compile time itself. This is called early binding or static binding or static linking also known as compile time polymorphism early binding means that an object is bound to its function call at compile time.



To select a particular member function while program is running is called run time polymorphism. C++ supports virtual functions to achieve this mechanism. Since the function is linked with a particular class much later after compilation, this process is called late binding. It is also called dynamic binding because selection of the appropriate function is done dynamically at run time.

Compile time :

```
int volume (int);
double volume (double, int);
long volume (long, int, int);
Run time polymorphism
class base
{
        public :
        virtual void show ( )
{
        cout << "show base class";}}.
```

Q. 6. (a) What is a-file mode? Discuss the various file mode options available in C++.

Ans. File Modes :

It specify the purpose for which the file is being opened. The syntax is as follows :

stream-object.open ("filename", mode);

The prototype of the class member functions contains only default values for the second argument and therefore they use the default values in the absence of the actual values. The default values are as follows :

IOS : in for jf stream function meaning open for reading only.

IOS : : out for of stream functions meaning open for writing only.

The other types of file nodes are as follows :

1. IOS:app – Append to end of file.
2. IOS : ate – go to end of file on opening.
3. IOS :: Binary – Binary file

4. IOS : in – Open file for reading only

5. IOS : no create – Open fails if the file does not exists.

6. IOS : noreplace – Open fails if the file already exists.

7. IOS : out – Open file for writing only

8. IOS : trunc – Delete contents of file if exists.

Q. 6. (b) What is a stream manipulator? Discuss the following manipulators : Setw ( ), setprecision ( ), setfill( ), setiosflags ( ).

Ans. Manipulators : These are operators that are used to format the data display the most commonly used manipulators are end and sent.

Manipulators are as follows :

Set w ( ) : It sets the width of output on console. The syntax is as follows :

Set w (int w)

```
cout <<setw (10) <<12345
```

This statement prints the value 12345 right justified in a field width of 10 characters.

Setprecision (int d) set the following point precision to d.

One statement can be used to format output for two or more values. e.g.

```
cout << set w (5) << set precision (2) <<12345
```

or

```
cout << set precision (4);
```

Set fill ( ) = set the fill character

The syntax for this manipulator is

set fill (int c) : it set the fill character to C. This is equivalent to the function fill ( ).

Setiosflags ( ): The syntax for this manipulator is

setiosflags (long f) and

it is used to set the format flag f.

The syntax for this in formatting is

```
cout << set w (15) <<set ios flags (ios :: scientific) << sqrt (3).
```

Q. 7. What do you understand by generic programming? What is a function template? Write a function template for sorting arrays of various types.

Ans. Generic programming is an approach where generic types are used as parameters in the algorithm so that they work for a variety of suitable data type and data structures. A template can be used to create a family of classes and functions. A template can be considered as a kind of macro.

Function Templates :

Function templates can be used to create a family of functions with different argument types. The general format of a function template is,

```
template <class T>
return type functionname (arg.of type T)
{
    //      Body of function with type T
            where ever appropriate
}
            function template swap ( ) is as follows :
            template <class T>
    void swap (T & x, T & y)
{
    T temp = x;
            x = y; y = temp; }
    Bubble sort using template function
    # include <iostream.n>
    template <class T>
    void bubble (T a [], int n)
{
    for (int i = 0; i <n –1; i++)
```

```
        for (int j = n–1; i < j; j– –)
        if (a [j] < a [j–1])
{
        swap (a [j], a [j–1]);}
}
        Template <class x>
        void swap (X & a, X & b)
{
        X temp = a;
        a = b, b = temp;
}
        int main ( )
{
        int x[5] ={10, 50, 30, 40, 20};
        float y [5] ={1.1, 5.5, 3.3, 4.4, 2.2};
        bubble (x, 5);
        bubble (y, 5);
        cout << "sorted X-array";
        for (int i = 0; i < 5; i++)
        cout << x [i];
        cout << end 1;
        cout << "sorted y-array :";
        for (int j =0; j < 5; j ++)
        cout << y [j];
        cout << end 1;
```

```
    return 0;
}
```

Q. 8. What does the inheritance mean in C++? What id containership? How does it differ from inheritance? Explain.

Ans. Inheritance :

The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as the base class and the new one is called derived class or subclass.

Containership :

C++ supports another way of inheriting properties of one class into another. This approach takes a view that an object can be a collection of many other objects. That is class can contain objects of other classes as its member shown below.

eg. :

```
class alph { ..... };
class beta {.....};
class gamma
{
    alpha a;
    beta b;
};
```

Creation of an object that contains another object is very different than the creation of an independent object. An independent object is created by its constructor when it is declared with arguments. On other hand containership or nested object is created in two stages :

1. The member objects are created using their respective constructors and then the,
2. Ordinary members are created.

This means constructions of all the member objects should be called before its own constructor body is executed. This is accomplished using an initializations list in the constructor of nested classes.