Lecture Notes

on

Object Oriented Programming

3CS4-06

Unit V

Department of Computer Science & Engineering
Jaipur Engineering College & Research Centre, Jaipur

Vision of the Institute

To become a renowned centre of outcome based learning and work toward academic, professional, cultural and social enrichment of the lives of individuals and communities.

Mission of the Institute

M1: Focus on evaluation of learning outcomes and motivate students to inculcate research aptitude by project based learning.
M2: Identify, based on informed perception of Indian, regional and global needs, the areas of focus and provide platform to gain knowledge and solutions.
M3: Offer opportunities for interaction between academia and industry.
M4: Develop human potential to its fullest extent so that intellectually capable and imaginatively gifted leaders can emerge in a range of professions.

Vision of the Department

To become renowned Centre of excellence in computer science and engineering and make competent engineers & professionals with high ethical values prepared for lifelong learning.

Mission of the Department

M1: To impart outcome based education for emerging technologies in the field of computer science and engineering.
M2: To provide opportunities for interaction between academia and industry.
M3: To provide platform for lifelong learning by accepting the change in technologies.
M4: To develop aptitude of fulfilling social responsibilities.

Program Outcomes (PO)

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and Computer Science & Engineering specialization to the solution of complex Computer Science & Engineering problems.
2. **Problem analysis**: Identify, formulate, research literature, and analyze complex Computer Science and Engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex Computer Science and Engineeringproblems and design system components or processes that meet the specified needs with appropriateconsideration for the public health and safety,and the cultural, societal, and environmentalconsiderations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of Computer Science and Engineering experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex Computer Science Engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional Computer Science and Engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional Computer Science and Engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the Computer Science and Engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings in Computer Science and Engineering.
10. **Communication**: Communicate effectively on complex Computer Science and Engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the Computer Science and Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation andability to engage in independent and life-long learning in the broadest contextof technological changein Computer Science and Engineering.

Program Educational Objectives (PEO)

PEO1: To provide students with the fundamentals of Engineering Sciences with more emphasis in Computer Science & Engineering by way of analyzing and exploiting engineering challenges.
PEO2: To train students with good scientific and engineering knowledge so as to comprehend, analyze, design, and create novel products and solutions for the real life problems inComputer Science and Engineering
PEO3: To inculcate professional and ethical attitude, effective communication skills, teamwork skills, multidisciplinary approach, entrepreneurial thinking and an ability to relate engineering issues with social issues for Computer Science & Engineering.
PEO4: To provide students with an academic environment aware of excellence, leadership, written ethical codes and guidelines, and the self-motivated life-long learning needed for a successful professional career in Computer Science & Engineering.
PEO5: To prepare students to excel in Industry and Higher education by Educating Students along with High moral values and Knowledge in Computer Science & Engineering.

Course Outcomes

1. Understand the paradigms of object oriented programming in comparison of procedural oriented programming.
2. Apply the class structure as fundamental, building block for computational programming.
3. Apply the major object-oriented concepts to implement object oriented programs in C++.
4. Implement the concept of abstraction inheritance, polymorphism, dynamic binding and generic structure in building reusable code.

Mapping of Course Outcomes with Program Outcomes

H=3, M=2, L=1

Semester	Subject	Code	L / T / P	CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
III	Object Oriented Programming	3CS4-06	L	Understand the paradigms of object oriented programming in comparison of procedural oriented programming.	3	3	3	2	2	2	1	1	0	1	1	3
			L	Apply the class structure as fundamental, building block for computational programming.	3	3	3	3	2	2	1	1	1	2	1	3
			L	Apply the major object-oriented concepts to implement object oriented programs in C++.	3	3	3	3	2	1	2	2	1	2	1	3
			L	Implement the concept of abstraction inheritance, polymorphism, dynamic binding and generic structure in building reusable code.	3	3	3	2	1	1	1	1	1	1	1	3

Syllabus

RAJASTHAN TECHNICAL UNIVERSITY, KOTA

Syllabus

II Year-III Semester: B.Tech. Computer Science and Engineering

3CS4-06: Object Oriented Programming

Credit-3
3L+0T+0P

Max. Marks : 150 (IA:30,ETE:120)
End Term Exam: 3 Hours

SN	CONTENTS	Hours
1	Introduction to different programming paradigm, characteristics of OOP, Class, Object, data member, member function, structures in C++, different access specifiers, defining member function inside and outside class, array of objects.	8
2	Concept of reference, dynamic memory allocation using new and delete operators, inline functions, function overloading, function with default arguments, constructors and destructors, friend function and classes, using this pointer.	8
3	Inheritance, types of inheritance, multiple inheritance, virtual base class, function overriding, abstract class and pure virtual function	9
4	Constant data member and member function, static data member and member function, polymorphism, operator overloading, dynamic binding and virtual function	9
5	Exception handling, Template, Stream class, File handling.	6
	TOTAL	40

UNIT V

Exception Handling:

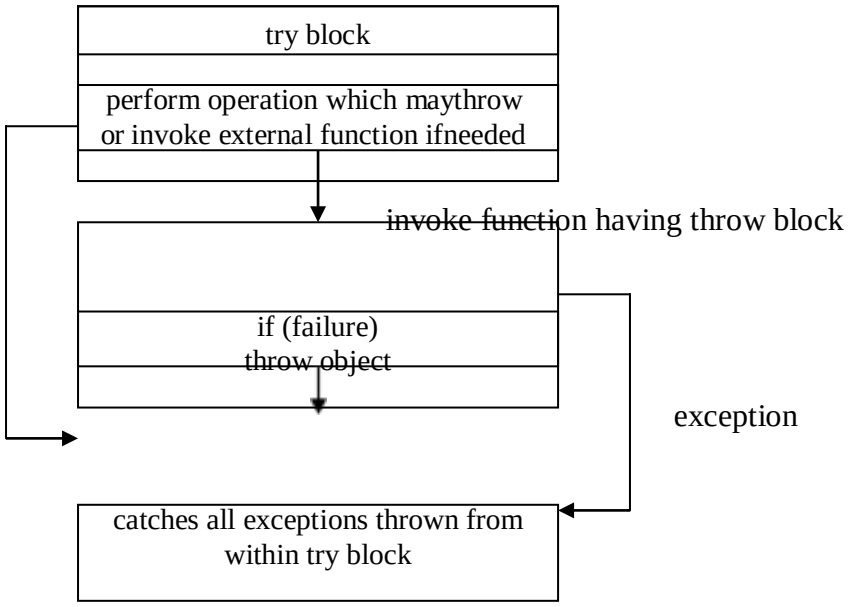
Exception refers to unexpected condition in a program. The unusual conditions could be faults, causing an error which in turn causes the program to fail. The error handling mechanism of c++ is generally referred to as exception handling.

Generally , exceptions are classified into synchronous and asynchronous exceptions.. The exceptions which occur during the program execution, due to some fault in the input data or technique that is not suitable to handle the current class of data. with in a program is known as synchronous exception.
Example:
errors such as out of range,overflow,underflow and so on.

The exceptions caused by events or faults unrelated to the program and beyond the control of program are asynchronous exceptions.
For example, errors such as keyboard interrupts, hardware malfunctions, disk failure and so on.

exception handling model:

When a program encounters an abnormal situation for which it in not designed, the user may transfer control to some other part of the program that is designed to deal with the problem. This is done by throwing an exception. The exception handling mechanism uses three blocks: try, throw and catch.
The try block must be followed immediately by a handler, which is a catch block. If an exception is thrown in the try block the program control is transferred to the appropriate exception handler. The program should attempt to catch any exception that is thrown by any function. The relationship of these three exceptions handling constructs called the exception handling model is shown in figure:



throw construct:

The keyword throw is used to raise an exception when an error is generated in the comutation. the throw expression initialize a temporary object of the typeT used in thorw (T arg).
syntax:

```
        throw T;
```

catch construct:

The exception handler is indicated by the catch keyword. It must be used immediately after the statements marked by the try keyword. The catch handler can also occur immediately after another catch Each handler will only evaluate an exception that matches.
syn:

```
catch(T)
{
// error meassges
}
```

try construct:
The try keyboard defines a boundary within which an exception can occur. A block of code in which an exception can occur must be prefixed by the keyword try. Following the try keyword is a block of code enclosed by braces. This indicates that the prepared to test for the existence of exceptions. If an exception occurs, the program flow is interrupted.

```
try
{
 …
if (failure)
        throw T;
}
catch(T)
{
…
}
```

example:

```
#include<iostream.h>
void main()
{
int a,b;
cout<<"enter two numbers:";
cin>>a>>b;
try
{
if (b= =0)
        throw b;
else
        cout<a/b;
}
catch(int x)
{
cout<<"2nd operand can't be 0";
}
}
```

Array reference out of bound:

```
#define max 5
class array
{
private:
        int a[max];
public:
        int &operator[](int i)
        {
        if (i<0 || i>=max)
                throw i;


        else
                return a[i];
        }
};
void main()
{
array x;
try
{
cout<<"trying to refer a[1]…"
x[1]=3;
cout<<"trying to refer a[13]…"
x[13]=5;
}
catch(int i)
{
cout<<"out of range in array references…";
}
}
```

multiple catches in a program

```
void test(int x)
{
try{
if (x==1)
        throw x;
else if (x==-1)
        throw 3.4;
else if (x==0)
        throw 's';
}
catch (int i)
{
cout<<"caught an integer…";
}
catch (float s)
{
cout<<"caught a float…";
```

```
}
catch (char c)
{
cout<<"caught a character…";
}}
void main()
{
test(1);
test(-1);
test(0);
}
```

catch all

```
void test(int x)
{
try{
if (x==1)
        throw x;
else if (x==-1)
        throw 3.4;
else if (x==0)
        throw 's';
}
catch (…)
{
cout<<"caught an error…";
}
```

Containership in C++

When a class contains objects of another class or its members, this kind of relationship is called containership or nesting and the class which contains objects of another class as its members is called as container class.

Syntax for the declaration of another class is:

```
Class class_name1
{
_____
_____
};
Class class_name2
{
_____
_____
};
Class class_name3
{
Class_name1obj1;                // object ofclass_name1
Class_name2obj2;                // object ofclass_name2
_____-
_____
};
```

```
//Sample Program to demonstrate Containership
#include < iostream.h >
#include < conio.h >
#include < iomanip.h >
#include< stdio.h >
const int len=80;
class employee
{
private:
char name[len];
int number;
public:
void get_data()
{
cout << "\n Enter employee name: ";
cin >> name;
cout << "\n Enter employee number:";
cin >>number;
}
void put_data()
{
cout << " \n\n Employee name: " << name;
cout << " \n\n Employee number: " <<number;
}
};
class manager
{
private:
char dept[len];
int numemp;
employee emp;
public:
void get_data()
{
emp.get_data();
cout << " \n Enter department: ";
cin >> dept;
cout << "\n Enter number of employees: ";
cin >> numemp;
}
void put_data()
{
emp.put_data();
cout << " \n\n Department: " << dept;
cout << " \n\n Number of employees: " << numemp;
}
};
class scientist
{
private:
int pubs,year;
employee emp;
public:
```

```
void get_data()
{
emp.get_data();
cout << " \n Number of publications: ";
cin >>pubs;
cout << " \n Year of publication: ";
cin >> year;
}
void put_data()
{
emp.put_data();
cout << "\n\n Number of publications: " << pubs;
cout << "\n\n Year of publication: "<< year;
}
};
void main()
{
manager m1;
scientist s1;
int ch;
clrscr();
do
{
cout << "\n 1.manager\n 2.scientist\n";
cout << "\n Enter your choice: ";
cin >> ch;
switch(ch)
{
case 1:
      cout << "\n Manager data:\n";
      m1.get_data();
      cout << "\n Manager data:\n";
      m1.put_data();
      break;
case 2:cout << " \n Scientist data:\n";
      s1.get_data();
      cout << " \n Scientistdata:\n";
      s1.put_data();
      break;
}
cout << "\n\n To continue Press 1 -> ";
cin >> ch;
}
while(ch==1);
getch();
}
```

Difference between Inheritance and Containership :

Containership: Containership is the phenomenon of using one or more classes within the definition of other class. When a class contains the definition of some other classes, it is referred to as composition, containment or aggregation. The data member of a new class is an object of some other class. Thus the other class is said to be composed of other classes and hence referred to as containership. Composition is often referred to as a "has-a" relationship because the objects of the composite class have objects of the composed class asmembers.

Inheritance: Inheritance is the phenomenon of deriving a new class from an old one. Inheritance supports code reusability. Additional features can be added to a class by deriving a class from it and then by adding new features to it. Class once written or tested need not be rewritten or redefined. Inheritance is also referred to as specialization or derivation, as one class is inherited or derived from the other. It is also termed as "is-a" relationship because every object of the class being defined is also an object of the inherited class.

Template:

Template supports generic programming, which allows developing reusable software components such as functions, classes, etc supporting different data types in a single frame work.

A template in c++ allows the construction of a family of template functions and classes to perform the same operation o different data types. The templates declared for functions are called class templates. They perform appropriate operations depending on the data type of the parameters passed tothem.

Function Templates:

A function template specifies how an individual function can be constructed.

```
template <class T>
return type functionnm(T arg1,T arg2)
{
fn body;
}
```

For example:

Input two number and swap their values

```
template <class T>
void swap (T &x,T & y)
{
T  z;
z=x;
x=y;
y=z;
}
void main( )
{
char ch1,ch2;
cout<<"enter two characters:";
cin>>ch1>>ch2;
swap(ch1,ch2);
cout<<ch1<<ch2;
int a,b;
cout<<"enter a,b:";
cin>>a>>b;
swap(a,b);
cout<<a<<b;
float p,q;
cout<<"enter p,q:";
cin>>p>>q;
swap(p,q);
cout<<p<<q;
}
```

example 2:

find maxium between two data items.

```
template <class T>
T max(T a,T b)
```

```
{
if (a>b)
return a;
else
return b;
}
void main()
{
char ch1,ch2;
cout<<"enter two characters:";
cin>>ch1>>ch2;
cout<<max(ch1,ch2);
int a,b;
cout<<"enter a,b:";
cin>>a>>b;
cout<<max(a,b);
float p,q;
cout<<"enter p,q:";
cin>>p>>q;
cout<<max(p,q);
}
```

Overloading of function template

```
#include<iostream.h>
template <class T>
void print( T a)
{
        cout<<a;
}
template <class T>
void print( T a, int n)
{
int i;
for(i=0;i<n;i++)
        cout<<a;
}
void main()
{
print(1);
print(3.4);
print(455,3);
print("hello",3);
}
```

Multiple arguments function template:

find sum of two different numbers

```
template <class T,class U>
T sum(T a,U b)
{
        return a+(U)b;
}
void main( )
```

```
{
cout<<sum(4,5.5);
cout<sum(5.4,3);
}
```

Class Template

similar to functions, classes can also be declared to operate on different data types. Such classes are class templates. a class template specifies how individual classes can be constructed similar to normal class definition. These classes model a generic class which support similar operations for different datatypes.

syn:

```
template <class T>
class classnm
{
T member1;
T member2;
…
…
public:
T fun();
…
..
};
```

objects for class template is created like:

```
classnm <datatype> obj;
obj.memberfun();
```

example:

Input n numbers into an array and print the element is ascending order.(array sorting)

```
template <class T>
class array
{
T *a;
int n;
public:
void getdata()
{
int i;
cout<<"enter how many no:";
cin>>n;
a=new T[n];
for (i=0;i<n;i++)
{
cout<<enter a number:";
cin>>a[i];
}
}
void putdata()
{
```

```
for (i=0;i<n;i++)
{
cout<<a[i]<<endl;
}
}
void sort( )
{
 T k;
int i,j;
for(i=0;i<n-1;i++)
{
        for (j=0;j<n;j++)
        {
                if (a[i]>a[j])
                        {
                                k=a[i];
                                a[i]=a[j];
                                a[j]=k;
                        }
        }
}
}
};
voidmain()
{
array <int>x;
x.getdata();
x.sort();
x.putdata();

array <float>y;
y.getdata():
y.sort();
y.p utdata();
}
```

Managing Console I/O

Introduction

One of the most essential features of interactive programming is its ability to interact with the users through operator console usually comprising keyboard and monitor. Accordingly, every computer language (and compiler) provides standard
input/output functions and/or methods to facilitate console operations.

C++ accomplishes input/output operations using concept of stream. A stream is a series of bytes whose value depends on the variable in which it is stored. This way, C++ is able to treat all the input and output operations in a uniform manner. Thus, whether it is reading from a file or from the keyboard, for a C++ program it is simply astream.

We have used the objects cin and cout (pre-defined in the iostream.h file) for the input and output of data of various types. This has been made possible by overloading the operators >> and << to recognize all the basic C++ types. The >>operator is overloaded in the istream class and « is overloaded in the ostream class.The
following is the general format for reading data from the keyboard:

cin >> variable1 >> variable2 >>… …>> variableN;

Where variable1, variable2, are valid C++ variable names that have been declared already. This statement will cause the computer to halt the execution and look for input data from the keyboard. The input data for this statement would be:

data1 data2. dataN

The input data are separated by white spaces and should match the type of variable in the cin list. Spaces, newlines and tabs will be skipped.

The operator >> reads the data character by character and assigns it to the indicated location. The reading for a variable will be terminated at the encounter of a white space or a character that does not match the destinationtype.

For example, consider the following code:

```
int code;
cin >> code;
```

Suppose the following data is given as input:

1267E

The operator will read the characters up to 7 and the value 1267 is assigned to code. The character E remains in the input stream and will be input to the next cin statement. The general format of outputtingdata:

cout << iteml <<item2 <<<< itemN;

The items, item1 through itemN may be variables or constants of any basic types.

The put() and get() Functions

The classes istream and ostream define two member functions get() and put() respectively to handle the single character input/output operations. There are two types of get() functions. We can use both get(char*) and get(void) prototypes to fetch a character including the blank space, tab and the newline character. The get(char*) version assigns the input character to its argument and the get(void) version returns the input character.

Since these functions are members of the input/output stream classes, we must invoke them using an appropriate object. For instance, look at the code snippet given below:

```
char c;
cin.get (c); //get a character from keyboard and assign it to c
while (c!= '\n')
{
        cout<<C;        //display the character on screen cin.get(c);
                        //get another character
}
```

This code reads and displays a line of text (terminated by a newline character). Remember, the operator>>can also be used to read a character but it will skip the white spaces and newline character. The above while loop will not work properly if the statement

```
cin >> c;
```

is used in place of

```
cin.get (c);
```

Try using both of them and compare the results. The get(void) version is used as follows:

```
char c;
c-cin.getl);      //cin.get (c)replaced
```

The value returned by the function get() is assigned to the variable c.

The function put(), a member of ostream class, can be used to output a line of text, character by character. For example,

cout << put ('x');
displays the character xand
cout << put (ch);
displays the value of variable ch.

The variable ch must contain a character value. We can also use a number as an argument to the function put (). For example,

cout << put (68) ;

displays the character D. This statement will convert the int value 90 to a char value and display the character whose ASCII value is 68,

The following segment of a program reads a line of text from the keyboard and displays it on the screen.

```
char c;.
cin.get(c)        //read a character
while (c!='\n')
{
     cout<< put(c); //display the character on screen cin.get (c ) ;
}
```

The getline () and write () Functions

We can read and display a line of text more efficiently using the line-oriented input/output functions getline() and write(). The getline() function reads a whole line of text that ends with a newline character. This function can be invoked by using the object cin as follows:

cin.getline(line, size);

This function call invokes the function which reads character input into the variable line. The reading is terminated as soon as either the newline character '\n' is encountered or size number of characters are read (whichever occurs first). The newline. character is read but not saved. Instead, it is replaced by the nullcharacter.

For example; consider the following code:

char name [20] ;
cin.getline(name, 20);

Assume that we have given the following input through the keyboard:

Neeraj good

This input will be read correctly and assigned to the character array name. Let us suppose the input is as follows:

Object Oriented Programming

In this case, the input will be terminated after reading the following 19 characters:

Object Oriented Pro

After reading the string/ cin automatically adds the terminating null character to thecharacter array.

Remember,thetwoblankspacescontainedinthestringarealsotakenintoaccount,i.e. between Objects and Oriented and Pro.

We can also read strings using the operator >>as follows:

cin >> name;

But remember cin can read strings that do not contain white space. This means that cin can read just one word and not a series of words such as “Neeraj good”.

Formatted Console I/O Operations

C++ supports a number of features that could be used for formattingtheoutput. These featuresinclude:

- ios class functions andflags.
- Manipulators.
- User-defined outputfunctions.

The ios class contains a large number of member functions that could be used to format the output in a number of ways. The most important ones among them are listed below.

Table 10.1

Function	Task
width()	To specify the required field size for displaying an output value
Precision()	To specify the number of digits to be displayed after the decimal point of a float value
fill()	To specify a character that is used to fill the unused portion of a field.
self()	To specify format flags that can control the form of output display (such as Left-justification and right-justification).
Unself()	To clear the flags specified.

Manipulators are special functions that can be included in these statements to alter the format parameters of a stream. The table given below shows some important! manipulator functions that are frequently used. To access these manipulators, the file iomanip.h should be included in the program.

Table 10.2

Manipulator	Equivalent Ios function
setw()	width()
Setprecision()	Precision()
Setfill()	fill()
setiosflags()	self()
Resetiosflags()	Unself()

In addition to these functions supported by the C++ library, we can create our own manipulator functions to provide any special ouput formats.