TITLE OF THE COURSE				
Course Code	21AMA391		**Credits**	2
Hours/Week (L-T-P-S)	3-0-0		**CIE Marks**	50
Total Teaching Hours	40		**SEE Marks**	100
Exam Hours	03		**Course Type**	Theory/ Integrated
Course Component	Application Development using C++			

COURSE LEARNING OUTCOMES

Students will be able to:

1. Design classes for the given application scenarios and apply the concepts of classes and objects, friend functions, friend classes, static class members and this pointer in implementation of solutions to problems.
2. Analyse and apply the concepts of function overloading, default arguments, constructors and operator overloading in designing and implementing solutions to given problems.
3. Apply the principle of code reusability using Inheritance, use the concept of virtual base classes to prevent the possible ambiguity in multiple inheritance and analyze the benefit of dynamic binding using virtual functions and inheritance.
4. Design and use template classes that take generic parameters to support code reusability principle in problem solving, understand and apply the concept of exception handling to prevent run-time abnormal program termination error in applications
5. Create text and binary files and use them in the applications to handle voluminous data.

COURSE CONTENTS

UNIT -1- (8 Hrs)

Object Oriented Programming Principles, The origins of C++, C++ Fundamentals- Introducing C++ Classes, Procedure Oriented Programming vs Object Oriented Programming, Class Diagram**.**

Classes and Objects: C++ Classes and Structures, Class and object declarations, Friend Functions, Friend Classes, Inline Functions, Static Class Members, The Scope Resolution Operator, Functions and Objects.

Arrays, Pointers, References, and the Dynamic Allocation Operators: Arrays of Objects, this pointer, References, C++'s Dynamic Allocation operators. Best practices and coding standards.

UNIT -2- (8 Hrs)

Function Overloading and Default Arguments: Function Overloading, Constructor types, Overloading Constructor Functions, Default Function Arguments, Function Overloading and Ambiguity

Operator Overloading: Creating a Member Operator Function, Operator Overloading Using a Friend Function, Namespaces.

UNIT -3- (8 Hrs)
Inheritance: Base-Class Access Control, Inheritance and Protected Members, Inheriting Multiple Base Classes, Constructors, Destructors and Inheritance, Granting Access, Virtual Base Classes Virtual Functions and Polymorphism: Virtual Functions, The Virtual Attribute Is Inherited, Virtual Functions Are Hierarchical, Pure Virtual Functions, Using Virtual Functions, Early vs Late Binding.
UNIT -4- (8 Hrs)
Templates: Generic Functions, Applying Generic Functions, Generic Classes Exception Handling: Exception Handling Fundamentals, Applying Exception Handling. Introduction to Standard Template Libraries: Introduction to STL Components, Containers, Algorithms, Iterators.
UNIT -5- (8 Hrs)
C++ File I/O: <fstream> and File Classes, Opening and Closing a File, Reading and Writing Text Files, Unformatted and Binary I/O, Detecting EOF, Random Access. Smart Pointers: auto pointer, RTTI

TEXTBOOKS					
SINO	**Unit**	**Textbook Title**	**Author(s)**	**Publisher(s)**	**Edition/Year of Publication**
1.	All	The Complete Reference C++	Herbert Schildt	TMH	4th Edition/2005
REFERENCE BOOKS					
1	All	C++	Primer, Stanley B. Lippman, Josee Lajoie, Barbara E. Moo	AddisonWesley	4th Edition/2005
2	All	Object-Oriented Programming with C++	Sourav Sahay	Oxford University Press	Edition/2006

ONLINE RESOURCES	
Topic/Title	**Link**
Introduction to Composite Materials	http://nptel.ac.in/course/112104168/1
Testing of Composite	http://nptel.ac.in/courses/112104221/20
Fracture and Safety of composites	http://nptel.ac.in/courses/101104010/20

COURSE ASSESSMENT METHOD
Continuous Internal Evaluation (CIE): • Three internals (Average of best of two) 30 Marks

- Programming Assignments 10 Marks
- Rubrics for the evaluation of Programming Assignments 10 Marks

Total 50 marks

Semester End Examination (SEE):

- Final Exam will be conducted for 100 marks.

PEDAGOGY

	CO-PO-PSO MAPPING														
CO	**PO**												**PSO1**	**PS02**	**PSO3**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**			
1	3	3		2									3		
2	2	3		2									2		
3		2	3	1					2	2	2	2	3		3
4		2	3	1					2	2	2	2	3		3
5	2	2											1		
CL	2	2	3	2					2	2	2	2	2		3

LIST OF PROGRAMS

1. Write a program to calculate final velocity using the formula: v=u + a*t, with initial velocity, acceleration and time as input.

2. Write a program to swap two characters of different data types using function overloading concept.

3. Write a program to change the sign of an operands using unary operator overloading concept.

4.Write a program to add two complex numbers using binary operator overloading concept.

5.Write a program to find mean value of two integers using friend function concept.

6. Write a program to multiply and divide two different data type using inline function concept.

7.Write a program to Implement Matrix class with dynamic memory allocation and necessary methods. Give proper constructor, destructor, copy constructor, and overloading of assignment operator.

8.Write a program to enter the sale value and print the agent's commission using single inheritance.

9.Write a program to enter salary and output income tax and net salary using multiple inheritance concept.

10. Write a program to enter the unit reading and output the customer's telephone bill using hierarchical inheritance.

11.Write a program to find the grade of the students based on academic marks and sports using multilevel inheritance.

12.Write a program having student as an abstract class and create many derived classes such as Engineering, Medical etc from student's class. Create their objects and process them.

13. Write a program to count the words and characters in given text using virtual function.

14. Write a program to calculate net pay of employee using virtual base class concept.

15. Write a program to calculate division of two number with a try block to detect and throw an exception if the condition "divide-by-zero" occurs.

PROGRAM 1:

```
#include <iostream>
using namespace std;
int main() {
  float v,u,a,t ;
  cout<<"Enter initial velocity \n" ;
  cin>> u ;
  cout<<"Enter acceleration \n" ;
  cin>> a ;
  cout<<"Enter time \n";
  cin>> t ;
  v = u + a*t ;
  cout<<v ; ;
  return 0;
}
```

OUTPUT:

If we take u as 1.2, a as 2.2 and t as 3.2. The output will be:

Enter initial velocity

Enter accelaration

Enter time

8.24

PROGRAM 2:

```
#include<iostream>
using namespace std;
void swap(int ,float );
int main()
{
   int a;
   float b;
   cout<<"Enter the Two Numbers to Swap in C++: ";
   cin>>a>>b;
   cout<<"\nAfter Swapping of Two Numbers:";
   swap(a,b);
   return 0;
}
void swap(int x,float y)
{
 int z;
 z=x;
 x=y;
 y=z;
 cout<<" "<<x<<"   "<<y;
}
```

OUTPUT:

Enter the Two Numbers to Swap in C++: 8 6

After Swapping of Two Numbers: 6 8

PROGRAM 3:

```
#include <iostream>
```

```
using namespace std;
class Count {
 private:
 int value;
 public:
    // Constructor to initialize count to 5
    Count() : value(5) {}
    // Overload ++ when used as prefix
    void operator ++ () {
        ++value;
    }
    void display() {
        cout << "Count: " << value << endl;
  }
};
int main() {
    Count count1;
    // Call the "void operator ++ ()" function
    ++count1;

    count1.display();
    return 0;
}
```

OUTPUT:

Count: 6

PROGRAM 4:

```
#include <iostream>
using namespace std;
```

```
class Complex
{
   private:
     float real;
     float imag;
   public:
      Complex(): real(0), imag(0){ }
      void input()
      {
         cout << "Enter real and imaginary parts respectively: ";
         cin >> real;
         cin >> imag;
      }
      Complex operator + (Complex c2)
      {
         Complex temp;
         temp.real = real + c2.real;
         temp.imag = imag + c2.imag;
         return temp;
      }
      void output()
      {
         if(imag < 0)
            cout << "Output Complex number: "<< real << imag << "i";
         else
            cout << "Output Complex number: " << real << "+" << imag << "i";
      }
};
int main()
{
```

```
    Complex c1, c2, result;
    cout<<"Enter first complex number:\n";
    c1.input();
    cout<<"Enter second complex number:\n";
    c2.input();
    result = c1 + c2;
    result.output();
    return 0;
}
```

OUTPUT:

```
Enter first complex number:
Enter real and imaginary parts respectively:
2
8

Enter second complex number:
Enter real and imaginary parts respectively:
4
3
Output Complex number: 6+11i
```

PROGRAM 5:

```
#include<iostream>
using namespace std;
```

```
class  base
{
  int val1,val2;
  public:
      void get()
      {
        cout<<"\nEnter 1st value :: ";
        cin>>val1;
        cout<<"\nEnter 2nd value :: ";
        cin>>val2;
      }
      friend float mean(base ob);
};
float mean(base ob) {
  return float(ob.val1+ob.val2)/2;
}
int main()
{
   base obj;
   obj.get();
   cout<<"\nMean value is :: "<<mean(obj)<<"\n";
   return 0;
}
```

OUTPUT:

Enter 1st value :: 5
Enter 2nd value :: 8
Mean value is :: 6.5
Process returned 0

PROGRAM 6:

```
#include <iostream>
using namespace std;
```

```
class operation
{
 int a, b,mul;
 float div;
public:
    void get();
    void product();
    void division();
};
inline void operation ::get()
{
    cout << "Enter first value:";
    cin >> a;
    cout << "Enter second value:";
    cin >> b;
}
inline void operation ::product()
{
    mul = a * b;
    cout << "Product of two numbers: " << a * b << "\n";
}
inline void operation ::division()
{
    div = a / b;
    cout << "Division of two numbers: " << a / b << "\n";
}
int main()
{
    cout << "Program using inline function\n";
    operation s;
```

```
    s.get();
    s.product();
    s.division();
    return 0;
}
```

OUTPUT:

```
Enter first value: 45
Enter second value: 15
Product of two numbers: 675
Division of two numbers: 3
```

PROGRAM 7:

```
#include<iostream>
using namespace std;
```

```
class matrix
{
int **m;
int row, col;
public:
matrix()
{
row=col=0;
m=NULL;
}
matrix(int r ,int c);
~matrix();
void getmatrix();
void showmatrix();
matrix(matrix &m2); //copy constructor
matrix& operator=(matrix &m2);
};
matrix::~matrix()
{
for(int i=0;i<row;i++)
delete m[i];
delete m;
}
matrix::matrix(int r ,int c)
{
row = r;
col = c;
m = new int*[row];
for(int i=0;i<row;i++)
m[i]=new int[col];
```

```
}
matrix::matrix(matrix &m2)
{
cout<<"\nCopy constructor invoked…\n";
row = m2.row;
col = m2.col;
m = new int*[row];
for(int i=0;i<row;i++)
m[i]=new int[col];
for(int i=0;i<row;i++)
for(int j=0;j<row;j++)
m[i][j]=m2.m[i][j];
}
matrix& matrix::operator=(matrix &m2)
{
cout<<"\nAssignment Operator Overloading…\n";
row = m2.row;
col = m2.col;
m = new int*[row];
for(int i=0;i<row;i++)
m[i]=new int[col];
for(int i=0;i<row;i++)
for(int j=0;j<row;j++)
m[i][j]=m2.m[i][j];
return *this;
}
void matrix::getmatrix()
{
for(int i=0;i<row;i++)
for(int j=0; j<col; j++)
```

```
cin>>m[i][j];
}
void matrix::showmatrix()
{
for(int i=0;i<row;i++)
{
for(int j=0;j<col;j++)
cout<<"\t"<<m[i][j];
cout<<"\n";
}
}
int main()
{
int r,c;
cout<<"\nEnter rows and cols of the matrix…\n";
cin>>r>>c;
matrix m1(r,c);
cout<<"\nEnter the matrix elements one by one…";
m1.getmatrix();
cout<<"\nEntered matrix is…\n";
m1.showmatrix();
//invoking copy constructor
matrix m2=m1;
cout<<"\nResult of copy constructor is…\n";
m2.showmatrix();
matrix m3;
m3=m1;
cout<<"\nResult of assignment operator overloading…\n";
m3.showmatrix();
return 0;
```

}

OUTPUT:

Enter rows and cols of the matrix…

3

2

Enter the matrix elements one by one…

2

3

1

4

5

7

Entered matrix is…

2 3

1 4

5 7

Copy constructor invoked…

Result of copy constructor is…

2 3

1 4

5 7

Assignment Operator Overloading…

Result of assignment operator overloading…

2 3

1 4

5 7

PROGRAM 8:

```
#include<iostream>
using namespace std;
```

```
class Agent
{
protected:
float commissionRate;
public:
Agent(float rate):commissionRate(rate){}
float calculateCommission(float saleValue){
return commissionRate*saleValue;
}
};
class Salesperson:public Agent {
public:
Salesperson(float rate):Agent(rate){}
void printCommission(float saleValue){
float commission=calculateCommission(saleValue);
cout<<"Agent's commission for sale value $ "<<saleValue<<"is $"<<commission<<endl;
}
};
int main()
{
float commissionRate;
cout<<"Enter the commission rate for the salesperson(in decimal):";
cin>>commissionRate;
Salesperson salesAgent(commissionRate);
float saleValue;
cout<<"Enter the sale value:$";
cin>>saleValue;
salesAgent.printCommission(saleValue);
return 0;
}
```

OUTPUT:-

Enter the commission rate for the salesperson(in decimal):

10.5

Enter the sale value:$

10000

Agent's commission for sale value $ 10000 is $105000

PROGRAM 9:

```
#include <iostream>
using namespace std;
```

```
class NetS
{
private:
 double nt;//Net sallary
public:
NetS(double pr)
{
  this->nt=pr;
}
double getNetSallary()const
{
  return this->nt;
}
};
class TaxS
{
private:
double tx;
public:
//Constructor
TaxS(double s)
{
  this->tx=s;
}
//getTaxSallary
double getTaxSallary()const
{
  return this->tx;
}
};
```

```
//Inheritanc
class Sallary:public TaxS,public NetS
{
  private:
double sal;
public:
//Constructor
  Sallary(double a,double b):NetS(a),TaxS(a)
{
  this->sal=a+b;
}
//getSallary
double getSallary()const
{
  return this->sal;
}
//print to stream
friend ostream& operator<<(ostream& os,Sallary& sal)
{
  os<<sal.getSallary()<<"$";
  return os;
}
};
int main() {
  double a;
  double b;
  cout<<"Net Sallary: ";
  cin>>a;
  cout<<"Tax sallary:";
  cin>>b;
```

```
  Sallary sal(a,b);
  cout<<sal<<endl;
  return 0;
}
```

OUTPUT:

Net Sallary: 15000

Tax sallary:5000

20000$

PROGRAM 10:

```
#include<iostream>
class TelephoneBill {
```

```
protected:
double ratePerUnit;
public:
TelephoneBill(double rate):ratePerUnit(rate) {}
virtual double calculateBill(int units) {
return units * ratePerUnit;
}
};
class Residential : public TelephoneBill {
public:
Residential(double rate): TelephoneBill(rate) {}
double calculateBill(int units) override {
if(units<=100)
return ratePerUnit*units;
else
return ratePerUnit*100+0.6*ratePerUnit*(units-100);
}
};
class Business: public TelephoneBill {
public:
Business(double rate): TelephoneBill(rate) {}
double calculateBill(int units) override {
if(units<=50)
return ratePerUnit*units;
else if(units<=150)
return ratePerUnit*50+0.5* ratePerUnit*(units-50);
else
return ratePerUnit*50+0.5*ratePerUnit*100+0.4*ratePerUnit*(units-150);
}
};
```

```
int main()
{
double residentialRate=0.5;
double businessRate=0.7;
Residential res(residentialRate);
Business bus(businessRate);
int units;
char customerType;
std::cout<<"Entercustomer type (R for Residential,B for Business):";
std::cin>>customerType;
std::cout<<"Enter units consumed:";
std::cin>>units;
double totalBill;
if(customerType=='R'){
totalBill=res.calculateBill(units);
std::cout<<"Total bill for Residential customer:$"<< totalBill<<std::endl;
}
else if(customerType=='B') {
totalBill=bus.calculateBill(units);
std::cout<<"Total bill for Business customer:$"<< totalBill<<std::endl;
}else
{
std::cout<<"Invalid customer type entered!"<<std::endl;
}
return 0;
}
```

OUTPUT:

```
Enter customer type (R for Residential,B for Business): R
Enter units consumed:10
```

Total bill for Residential customer:$5

Enter customer type (R for Residential,B for Business): B

Enter units consumed: 20

Total bill for Business customer:$14

PROGRAM 11:

```
#include<iostream>
using namespace std;
```

```
class Academic {
protected:
float marks;
public:
Academic(float m):marks(m) {}
char calculateGrade() {
if(marks>=90)
return 'A';
else if(marks>=80)
return 'B';
else if(marks>=70)
return 'c';
else if(marks>=60)
return 'D';
else
return 'F';
}
};
class Sports:public Academic {
protected:
int sportsScore;
public:
Sports(float m,int s) :Academic(m),sportsScore(s) {}
char calculateGrade() {
char academicGrade = Academic::calculateGrade();
if(sportsScore>=90)
return (academicGrade=='A')?'S':'A';
else if(sportsScore>=80)
return (academicGrade=='A')?'A':'B';
else if(sportsScore>=70)
```

```
return (academicGrade=='F')?'F':'C';
else
return academicGrade;
}
};
class Student:public Sports {
public:
Student(float m,int s) :Sports(m,s) {}
void displayGrade() {
char grade=calculateGrade();
cout<<"AcademicMarks:"<<marks<<",Sports
Score:"<<sportsScore<<",Grade:"<<grade<<endl;
}
};
int main()
{
float academicMarks;
int sportsScore;
cout<<"Enter Academic Marks:";
cin>>academicMarks;
cout<<"Enter Sports Score:";
cin>>sportsScore;
Student student(academicMarks,sportsScore);
student.displayGrade();
return 0;
}
```

OUTPUT:-

```
Enter Academic Marks:
100
```

Enter Sports Score:

80

AcademicMarks:100,Sports Score:80,Grade:A

PROGRAM 12:

```
#include<iostream>
#include<string>
```

```
using namespace std;
class student
{
protected:
     string name;
     int age;
 public:
      virtual void display()=0;
      virtual void study()=0;
};
class EngineeringStudent:public student
{
public:
    EngineeringStudent(string n,int a){
    name=n;
    age=a;
    }
    void display() override {
    cout<<"Engineering Student:"<<name<<",Age:"<<age<<endl;
    }
    void study() override {
    cout<<"is studying engineering:"<<name<<",Age:"<<endl;
    }
    };
class MedicalStudent:public student
{
public:
   MedicalStudent(string n,int a){
   name=n;
   age=a;
```

```
    }
    void display() override {
    cout<<"Medical Student:"<<name<<",Age:"<<age<<endl;
    }
    void study() override {
    cout<<"is studying medical:"<<name<<",Age:"<<endl;
    }
    };
int main()
{
EngineeringStudent engStudent("John",20);
MedicalStudent medStudent("Emily",22);
engStudent.display();
engStudent.study();
medStudent.display();
medStudent.study();
return 0;
}
```

OUTPUT:-

```
Engineering Student:John,Age:20
is studying engineering:John,Age:
Medical Student:Emily,Age:22
is studying medical:Emily,Age:
```

PROGRAM 13:

```
#include<iostream>
#include<string>
```

```
class Counter {
public:
virtual int countWords(const std::string & text)=0;
virtual int countCharacters(const std::string & text)=0;
};
class WordCharacterCounter : public Counter {
public:
int countWords(const std::string & text)override {
int words=0;
bool inWord=false;
for(char c:text) {
if(std::isalpha(c) && !inWord){
inWord=true;
words++;
}
else if(!std::isalpha(c)){
inWord=false;
}
}
return words;
}
int countCharacters(const std::string& text) override {
return text.length();
}
};
int main()
{
WordCharacterCounter counter;
std::string input_text="This is a sample text.";
int word_count=counter.countWords(input_text);
```

```
int char_count=counter.countCharacters(input_text);
std::cout<<"Word count:"<<word_count<<std::endl;
std::cout<<"Character count:"<<char_count<<std::endl;
return 0;
}
```

OUTPUT:-

Word count:5

Character count:22

PROGRAM 14:

```
#include<iostream>
using namespace std;
```

```
class Employee {
protected:
float salary;
public:
Employee(float sal) : salary(sal) {}
virtual float calculateNetPay()
{
return salary;
}
};
class Deductions:virtual public Employee {
protected:
float deductions;
public:
Deductions(float sal,float ded) : Employee(sal),deductions(ded) {}
float calculateNetPay() override {
return salary-deductions;
}
};
class Allowances:virtual public Employee {
protected:
float allowances;
public:
Allowances(float sal,float allow) : Employee(sal),allowances(allow) {}
float calculateNetPay() override {
return salary+allowances;
}
};
class SalaryCalculator:public Deductions, public Allowances {
public:
```

```
SalaryCalculator(float sal,float ded,float allow) :
Employee(sal),Deductions(sal,ded),Allowances(sal,allow) {}
float calculateNetPay() override {
return Deductions::calculateNetPay()+Allowances::calculateNetPay()-salary;
}
};
int main()
{
float salary,deductions,allowances;
cout<<"Enter salary:$";
cin>>salary;
cout<<"Enter deductions:$";
cin>>deductions;
cout<<"Enter allowances:$";
cin>>allowances;
SalaryCalculator emp(salary,deductions,allowances);
float netPay=emp.calculateNetPay();
cout<<"Net pay of the employee is: $"<<netPay<<endl;
return 0;
}
```

OUTPUT:-

```
Enter salary:$
20000
Enter deductions:$
4000
Enter allowances:$
3000
Net pay of the employee is: $ 19000
```

PROGRAM 15:

```
#include<iostream>
using namespace std;
```

```
int main()
{
int var1,var2;
float var3;
cout<<"enter the dividend:";
cin>>var1;
cout<<"\n";
cout<<"enter the divisor:";
cin>>var2;
cout<<"\n";
//exception handling begins here
try //try block
{
if(var2!=0) //checking if divisor is zero
{
var3=var1/var2;
cout<<"outcome :"<<var3;
}
else
{
throw(var2); //throwing the exception found
}
}
//catch block
catch(int exc)
{
cout<<"division by zero is not possible. Please try again with different value of variables";
}
}
```

OUTPUT:

Enter the dividend:4

Enter the divisor:2

Outcome :2