List of C++ programs for Practical Exam

1. Create a C++ program to print following using manipulator functions: endl, setw( ), setfill ( ), setprecision( ) **(20 marks)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

| PES’s |

| Modern College Ganeshkhind |

| Computer Science Department |

| Subject : C++ |

| Roll No Percent Grade |

| 1 |

| 2 |

| 3 |

| 4 |

#include <iostream>

using namespace std;

#include<iomanip>

int main() {

    cout<<setfill('\*')<<setw(70);

    cout<<""<<endl;

    cout<<"|"<<setfill(' ')<<setw(30)<<"PES's"<<setfill(' ')<<setw(30)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(40)<<"Modern College Ganeshkhind"<<setfill(' ')<<setw(20)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(40)<<"Computer Science Department"<<setfill(' ')<<setw(20)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(30)<<"Subject : C++"<<setfill(' ')<<setw(30)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(30)<<"Roll No   Percent   Grade"<<setfill(' ')<<setw(30)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(10)<<"1"<<setfill(' ')<<setw(50)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(10)<<"2"<<setfill(' ')<<setw(50)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(10)<<"3"<<setfill(' ')<<setw(50)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(10)<<"4"<<setfill(' ')<<setw(50)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(10)<<"5"<<setfill(' ')<<setw(50)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(10)<<"6"<<setfill(' ')<<setw(50)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(10)<<"."<<setfill(' ')<<setw(50)<<"|"<<endl;

    cout<<"|"<<setfill(' ')<<setw(10)<<"."<<setfill(' ')<<setw(50)<<"|"<<endl;

    cout<<setfill('\*')<<setw(70);

    cout<<""<<endl;

    return 0;

}

1. Write a C++ program to create array of objects of Book (id, title, author, price, publication) class. Read number of objects n from the user. **(15 marks)**

**#include<iostream>**

**#include <string>**

**using namespace std;**

**class Book**

**{**

**int id;**

**string title;**

**string author;**

**float price;**

**string publication;**

**public:**

**void getdata();//Declaration of function**

**void putdata();//Declaration of function**

**};**

**void Book::getdata(){//Defining of function**

**cout<<"Enter Id : ";**

**cin>>id;**

**cout<<"Enter title : ";**

**cin>>title;**

**cout<<"Enter author : ";**

**cin>>author;**

**cout<<"Enter price : ";**

**cin>>price;**

**cout<<"Enter publication : ";**

**cin>>publication;**

**}**

**void Book::putdata(){//Defining of function**

**cout<<"id="<<id<<endl;**

**cout<<"title="<<title<<endl;**

**cout<<"author="<<author<<endl;**

**cout<<"price="<<price<<endl;**

**cout<<"publication="<<publication<<endl;**

**cout<<endl;**

**}**

**int main(){**

**//One member**

**int n;**

**cout<<"Enter the number of books:";**

**cin>>n;**

**Book emp[n];**

**for(int i=0;i<n;i++)**

**{**

**emp[i].getdata();//Accessing the function**

**}**

**for(int i=0;i<n;i++)**

**{**

**emp[i].putdata();//Accessing the function**

**}**

**return 0;**

**}**

1. [Write C++ program to count number of alphabets, digits and special characters in string](https://techstudy.org/CplusplusLanguage/Write-Cplusplus-program-to-count-number-of-alphabets-digits-and-special-characters-in-string) **(10 marks)**

#include <iostream>

#include <cctype>

using namespace std;

int main() {

    string str;

    int alphabets = 0, digits = 0, special\_chars = 0;

    // Read the string

    cout << "Enter a string: ";

    getline(cin, str);

    // Iterate through each character in the string

    for (char c : str) {

        if (isalpha(c)) {

            // If the character is an alphabet, increment the alphabet count

            alphabets++;

        } else if (isdigit(c)) {

            // If the character is a digit, increment the digit count

            digits++;

        } else {

            // If the character is neither an alphabet nor a digit, it is a special character

            special\_chars++;

        }

    }

    // Print the count of alphabets, digits, and special characters

    cout << "Number of alphabets: " << alphabets << endl;

    cout << "Number of digits: " << digits << endl;

    cout << "Number of special characters: " << special\_chars << endl;

    return 0;

}

    // find\_first\_not\_of function: searches for the first occurrence of a character not in the given string in the original string

    cout << "find\_first\_not\_of('orl', 3, 3): " << str.find\_first\_not\_of("orl", 3, 3) << endl;

    // find\_last\_of function: searches for the last occurrence of any character from the given string in the original string

    cout << "find\_last\_of('orl', 5, 3): " << str.find\_last\_of("orl", 5, 3) << endl;

    // find\_last\_not\_of function: searches for the last occurrence of a character not in the given string in the original string

    cout << "find\_last\_not\_of('orl', 5): " << str.find\_last\_not\_of("orl", 5) << endl;

    // push\_back function: adds the given character to the end of the string

    str.push\_back('!');

    cout << "push\_back('!'): " << str << endl;

    // pop\_back function: removes the last character from the string

    str.pop\_back();

    cout << "pop\_back(): " << str << endl;

    return 0;

}

1. Write a C++ program to implement following string functions:

* substr(int position, int length)
* append(const string& str1)
* find(string& str1, int position, int len)
* find\_first\_of(string& str1, int position, int len)
* find\_first\_not\_of(string& str1, int position, int n)
* find\_last\_of(string& str1, int position, int n)
* find\_last\_not\_of(string& str1, int position)
* push\_back(char c)
* pop\_back()

Read the string from the user. Show the output of each function with proper message. (any 3 functions **– 10 marks**)

#include <iostream>

#include <string>

using namespace std;

int main()

{

    // Read the string from the user

    string str;

    cout << "Enter a string: ";

    getline(cin, str);

    // substr function: returns a substring of the given string

    cout << "substr(2, 5): " << str.substr(2, 5) << endl;

    // append function: appends the given string to the end of the original string

    string str1 = " World";

    str.append(str1);

    cout << "append(' World'): " << str << endl;

    // find function: searches for the given string in the original string and returns its position

    cout << "find('orld', 3, 4): " << str.find("orld", 3, 4) << endl;

    // find\_first\_of function: searches for the first occurrence of any character from the given string in the original string

    cout << "find\_first\_of('orl', 3, 3): " << str.find\_first\_of("orl", 3, 3) << endl;

    // find\_first\_not\_of function: searches for the first occurrence of a character not in the given string in the original string

    cout << "find\_first\_not\_of('orl', 3, 3): " << str.find\_first\_not\_of("orl", 3, 3) << endl;

    // find\_last\_of function: searches for the last occurrence of any character from the given string in the original string

    cout << "find\_last\_of('orl', 5, 3): " << str.find\_last\_of("orl", 5, 3) << endl;

    // find\_last\_not\_of function: searches for the last occurrence of a character not in the given string in the original string

    cout << "find\_last\_not\_of('orl', 5): " << str.find\_last\_not\_of("orl", 5) << endl;

    // push\_back function: adds the given character to the end of the string

    str.push\_back('!');

    cout << "push\_back('!'): " << str << endl;

    // pop\_back function: removes the last character from the string

    str.pop\_back();

    cout << "pop\_back(): " << str << endl;

    return 0;

}

1. Write a C++ program to calculate Compound Interest using concept of pass by reference and return by reference. Read principal amount, rate of interest and period from the user. (**10 marks**)

#include <iostream>

#include <cmath>

using namespace std;

// Function prototype

void calculateCompoundInterest(double &principal, double &rate, int &period, double &compoundInterest);

int main()

{

    double principal, rate, compoundInterest;

    int period;

    // Read principal amount, rate of interest, and period from the user

    cout << "Enter principal amount: ";

    cin >> principal;

    cout << "Enter rate of interest: ";

    cin >> rate;

    cout << "Enter period (in years): ";

    cin >> period;

    // Calculate compound interest

    calculateCompoundInterest(principal, rate, period, compoundInterest);

    // Display the compound interest

    cout << "Compound interest: " << compoundInterest << endl;

    return 0;

}

// Function to calculate compound interest

void calculateCompoundInterest(double &principal, double &rate, int &period, double &compoundInterest)

{

    compoundInterest = principal \* pow(1 + (rate / 100), period) - principal;

}

1. Write a function Interest(principal, int\_rate, year). Use function overloading and default arguments to set the values of int\_rate and year. (**10 marks**)
2. #include <iostream>
3. using namespace std;
4. double Interest(double principal, double int\_rate, int year)
5. {
6. // Calculate the interest
7. double interest = principal \* int\_rate \* year;
8. // Return the interest
9. return interest;
10. }
11. // Overloaded function with default arguments
12. double Interest(double principal, double int\_rate = 0.05)
13. {
14. int year=1;
15. // Call the original function with the default arguments
16. return Interest(principal, int\_rate, year);
17. }
18. int main()
19. {
20. // Calculate the interest on 1000 at 5% for 1 year
21. double interest1 = Interest(1000);
22. cout << "Interest: " << interest1 << endl;
23. // Calculate the interest on 1000 at 10% for 2 years
24. double interest2 = Interest(1000, 0.1, 2);
25. cout << "Interest: " << interest2 << endl;
26. return 0;
27. }

OUTPUT

1. Create a class Cone(radius, height). Declare a static member count that will store the number of objects created. Write a static function that display() the number of objects created. Write an area() function as inline function to display area of the cone. (15 marks)

#include <iostream>

#include <cmath>

using namespace std;

class Cone

{

private:

    double radius;

    double height;

    static int count; // static member variable to store the number of objects created

public:

    // Constructor to initialize radius and height of the cone

    Cone(double r, double h)

    {

        radius = r;

        height = h;

        ++count; // increment the count each time an object is created

    }

    // static function to display the number of objects created

    static void display()

    {

        cout << "Number of objects created: " << count << endl;

    }

    // inline function to calculate and display the surface area of the cone

    inline double area()

    {

        //double surface\_area = M\_PI \* radius \* (radius + sqrt(pow(radius, 2) + pow(height, 2)));

        //cout << "Surface area of the cone: " << surface\_area << endl;

        //double surface\_area = M\_PI \* radius \* (radius + sqrt(pow(radius, 2) + pow(height, 2)));

       // return surface\_area;

    }

};

// initialize the static member variable

int Cone::count = 0;

int main()

{

    // create two Cone objects

    Cone cone1(2, 3);

    Cone cone2(4, 5);

    // display the number of objects created

    Cone::display(); // Output: Number of objects created: 2

    // calculate and display the surface area of the first cone

    cout<<"Surface area of cone1: "<<cone1.area(); // Output: Surface area of the cone: 37.699111843077515

    cout<<"Surface area of cone2: "<<cone2.area();

    return 0;

}

1. Create a class Account(Ano, Name, balance, int\_rate). Write a friend function Total\_bal() to calculate total interest earned on the balance. (10 marks)

#include <iostream>

#include <string>

using namespace std;

class Account

{

private:

    int A\_no;

    string Name;

    double balance;

    double int\_rate;

public:

    Account(int A, string N, double B, double IR) : A\_no(A), Name(N), balance(B), int\_rate(IR) {}

    friend double Total\_bal(const Account& a);

};

double Total\_bal(const Account& a)

{

    return a.balance \* a.int\_rate;

}

int main()

{

    Account a(12345, "John Smith", 1000.0, 0.05);

    cout << "Total interest: " << Total\_bal(a) << endl;

    return 0;

}

1. Create a class Manager(id, name, department, designation). Use default, parameterized and copy constructors to initialize the members of the class. (10 marks)

#include <iostream>

using namespace std;

class Manager

{

private:

    int id;

    string name;

    string department;

    string designation;

public:

    Manager()

        : id(0), name(""), department(""), designation("") {}

    Manager(int ID, string Name, string Department, string Designation)

        : id(ID), name(Name), department(Department), designation(Designation) {}

    Manager(const Manager& m)

        : id(m.id), name(m.name), department(m.department), designation(m.designation) {}

    void print()

    {

        cout << "ID: " << id << endl;

        cout << "Name: " << name << endl;

        cout << "Department: " << department << endl;

        cout << "Designation: " << designation << endl;

        cout << endl;

    }

};

int main()

{

    Manager m1; // Uses default constructor

    Manager m2(12345, "John Smith", "IT", "Manager"); // Uses parameterized constructor

    Manager m3(m2); // Uses copy constructor

    m1.print();

    m2.print();

    m3.print();

    return 0;

}

1. Write a program to overload arithmetic operators (+, -, / , \* ) using operator function as member and friend function. [Use class Point(int x,int y)] (20 marks) (only 3 operators)

#include <iostream>

using namespace std;

class Point

{

private:

    int x;

    int y;

public:

    Point(int X, int Y) : x(X), y(Y) {}

    // Member operator functions

    Point operator+(const Point& p)

    {

        return Point(x + p.x, y + p.y);

    }

    Point operator-(const Point& p)

    {

        return Point(x - p.x, y - p.y);

    }

    Point operator\*(const Point& p)

    {

        return Point(x \* p.x, y \* p.y);

    }

    Point operator/(const Point& p)

    {

        return Point(x / p.x, y / p.y);

    }

    void print()

    {

        cout << "(" << x << ", " << y << ")" << endl;

    }

};

int main()

{

    Point p1(1, 2);

    Point p2(3, 4);

    // Member operator functions

    Point p3 = p1 + p2; // Uses operator+

    p3.print();

    Point p4 = p1 - p2; // Uses operator-

    p4.print();

    Point p5 = p1 \* p2; // Uses operator\*

    p5.print();

    Point p6 = p1 / p2; // Uses operator/

    p6.print();

    return 0;

}

1. Write a program to overload Relational operators (>, <, >=, <= !=, == ) using operator function as member and friend function. [Use class Complex(int real, int img)] (20 marks) (only 3 operators)

#include <iostream>

class Complex

{

private:

    int real;

    int img;

public:

    Complex(int real = 0, int img = 0) : real(real), img(img) {}

     // Overloading the < operator as a friend function

    friend bool operator<(const Complex& c1, const Complex& c2)

    {

        // Compare the magnitudes of the two complex numbers

        return (c1.real \* c1.real + c1.img \* c1.img) < (c2.real \* c2.real + c2.img \* c2.img);

    }

    // Overloading the > operator as a friend function

    friend bool operator>(const Complex& c1, const Complex& c2)

    {

        // Compare the magnitudes of the two complex numbers

        return (c1.real \* c1.real + c1.img \* c1.img) > (c2.real \* c2.real + c2.img \* c2.img);

    }

    // Overloading the <= operator as a friend function

    friend bool operator<=(const Complex& c1, const Complex& c2)

    {

        // Compare the magnitudes of the two complex numbers

        return (c1.real \* c1.real + c1.img \* c1.img) <= (c2.real \* c2.real + c2.img \* c2.img);

    }

    // Overloading the >= operator as a friend function

    friend bool operator>=(const Complex& c1, const Complex& c2)

    {

        // Compare the magnitudes of the two complex numbers

        return (c1.real \* c1.real + c1.img \* c1.img) >= (c2.real \* c2.real + c2.img \* c2.img);

    }

// Overloading the == operator as a friend function

    friend bool operator==(const Complex& c1, const Complex& c2)

    {

        // Check if the real and imaginary parts are equal

        return (c1.real == c2.real) && (c1.img == c2.img);

    }

// Overloading the != operator as a friend function

    friend bool operator!=(const Complex& c1, const Complex& c2)

    {

        // Check if the real and imaginary parts are equal

        return (c1.real != c2.real) || (c1.img != c2.img);

    }

};

int main()

{

    Complex c1(1, 2);

    Complex c2(2, 3);

    Complex c3(1, 2);

    std::cout << (c1 > c2) << std::endl;  // Output: 0

    std::cout << (c1 < c2) << std::endl;  // Output: 1

    std::cout << (c1 >= c2) << std::endl; // Output: 0

    std::cout << (c1 <= c2) << std::endl; // Output: 1

    std::cout << (c1 != c2) << std::endl; // Output: 1

    std::cout << (c1 == c2) << std::endl; // Output: 0

    return 0;

}

1. Write a program to overload Assignment operator (=). [Use class Result(int m1, int m2, int m3)] (15 marks)

#include <iostream>

class Result {

 public:

  // Constructor

  Result(int m1, int m2, int m3) : m1\_(m1), m2\_(m2), m3\_(m3) {}

  // Overloaded assignment operator

  Result& operator=(const Result& other) {

    m1\_ = other.m1\_;

    m2\_ = other.m2\_;

    m3\_ = other.m3\_;

    return \*this;

  }

  // Getters for the marks

  int m1() const { return m1\_; }

  int m2() const { return m2\_; }

  int m3() const { return m3\_; }

 private:

  int m1\_, m2\_, m3\_;

};

int main() {

  Result r1(50, 60, 70);

  Result r2(80, 90, 100);

  // Assign r2 to r1 using the overloaded assignment operator

  r1 = r2;

  std::cout << "r1.m1 = " << r1.m1() << std::endl;

  std::cout << "r1.m2 = " << r1.m2() << std::endl;

  std::cout << "r1.m3 = " << r1.m3() << std::endl;

  return 0;

}

1. Write a program to overload insertion and extraction operators (<<,>>). [Use class Distance(inch, feet)] (15 marks)

**#include <iostream>**

**using namespace std;**

**class Distance {**

**public:**

**int inch;**

**int feet;**

**// Constructor**

**Distance(int inch, int feet) : inch(inch), feet(feet) {}**

**Distance() : inch(0), feet(0) {}**

**// Overload the stream insertion operator (<<)**

**friend ostream& operator<<(ostream& out, const Distance& d) {**

**out << d.feet << "' " << d.inch << "\"";**

**return out;**

**}**

**// Overload the stream extraction operator (>>)**

**friend istream& operator>>(istream& in, Distance& d) {**

**in >> d.feet >> d.inch;**

**return in;**

**}**

**};**

**int main() {**

**// Create a Distance object**

**Distance d; //calls default constructor**

**// Use the overloaded stream insertion operator to print the Distance object**

**cout << d << endl;**

**cout<<"Enter feet and inches"<<endl;**

**// Use the overloaded stream extraction operator to read values for the Distance object**

**cin >> d;**

**// Use the overloaded stream insertion operator to print the modified Distance object**

**cout << d << endl;**

**return 0;**

**}**

1. Create a base class Employee(Name, Id). Inherit a class Manager(Department,Salary) from Employee. Inherit a class SalesManager(Incentive) from Manager. Use appropriate parameterized constructors for each class. (20 marks)

#include <iostream>

#include <string>

using namespace std;

class Employee {

 public:

  Employee(string name, int id) : name\_(name), id\_(id) {}

  string getName() const { return name\_; }

  int getId() const { return id\_; }

 private:

  string name\_;

  int id\_;

};

class Manager : public Employee {

 public:

  Manager(string name, int id, string department, int salary)

      : Employee(name, id), department\_(department), salary\_(salary) {}

  string getDepartment() const { return department\_; }

  int getSalary() const { return salary\_; }

 private:

  string department\_;

  int salary\_;

};

class SalesManager : public Manager {

 public:

  SalesManager(string name, int id, string department, int salary, int incentive)

      : Manager(name, id, department, salary), incentive\_(incentive) {}

  int getIncentive() const { return incentive\_; }

 private:

  int incentive\_;

};

int main() {

  Employee e("Alice", 12345);

  Manager m("Bob", 54321, "Marketing", 100000);

  SalesManager sm("Charlie", 67890, "Sales", 110000, 5000);

  cout << e.getName() << " (" << e.getId() << ")" << endl;

  cout << m.getName() << " (" << m.getId() << ") - " << m.getDepartment() << " - $" << m.getSalary() << endl;

  cout << sm.getName() << " (" << sm.getId() << ") - " << sm.getDepartment() << " - $" << sm.getSalary() << " - $" << sm.getIncentive() << " incentive" << endl;

  return 0;

}

1. Derive a class Result(Percentage,Grade) from two base classes namely Student(Rollno, Name) and Marks(Comp, Maths, Elec, Stats). Calculate the percentage using a member function calc\_per(), display the grade using disp\_grade() member function. Display three toppers out of n students.(20 marks)

#include <iostream>

#include <string>

class Student

{

  public:

    Student(int rollno, std::string name) : rollno\_(rollno), name\_(name) {}

    int rollno() const { return rollno\_; }

    std::string name() const { return name\_; }

  private:

    int rollno\_;

    std::string name\_;

};

class Marks

{

  public:

    Marks(int comp, int maths, int elec, int stats) : comp\_(comp), maths\_(maths), elec\_(elec), stats\_(stats) {}

    int comp() const { return comp\_; }

    int maths() const { return maths\_; }

    int elec() const { return elec\_; }

    int stats() const { return stats\_; }

  private:

    int comp\_;

    int maths\_;

    int elec\_;

    int stats\_;

};

class Result : public Student, public Marks

{

  public:

    Result(int rollno, std::string name, int comp, int maths, int elec, int stats)

        : Student(rollno, name), Marks(comp, maths, elec, stats) {}

    double calc\_per()

    {

        return (comp() + maths() + elec() + stats()) / 4.0;

    }

    std::string disp\_grade()

    {

        double percentage = calc\_per();

        if (percentage >= 90)

            return "A+";

        else if (percentage >= 80)

            return "A";

        else if (percentage >= 70)

            return "B+";

        else if (percentage >= 60)

            return "B";

        else if (percentage >= 50)

            return "C+";

        else if (percentage >= 40)

            return "C";

        else

            return "D";

    }

};

int main()

{

    // Create an array of Results for n students

    const int n = 5;

    Result students[n] = {

        Result(1, "Alice", 90, 85, 80, 95),

        Result(2, "Bob", 70, 75, 80, 85),

        Result(3, "Charlie", 60, 65, 70, 75),

        Result(4, "Dave", 50, 55, 60, 65),

        Result(5, "Eve", 40, 45, 50, 55)};

    // Find the top three students by sorting the array

    for (int i = 0; i < n - 1; i++)

    {

        for (int j = 0; j < n - i - 1; j++)

        {

            if (students[j].calc\_per() < students[j + 1].calc\_per())

            {

                std::swap(students[j], students[j + 1]);

            }

        }

    }

     // Print the top three students

  std::cout << "Top three students:" << std::endl;

  for (int i = 0; i < 3; i++) {

    std::cout << students[i].name() << ": " << students[i].calc\_per() << "%"

         << " (" << students[i].disp\_grade() << ")" <<std:: endl;

  }

  return 0;

}

1. Create a class Person(Name, Aadhar\_no). Derive classes Doctor(Speciality, place\_of\_hosp ) and Teacher(Subject, college\_name). Use appropriate constructors and destructors (15 marks)

#include <iostream>

#include <string>

using namespace std;

class Person {

 public:

  Person(string name, string aadhar\_no) : name\_(name), aadhar\_no\_(aadhar\_no) {}

  virtual ~Person() {}

  string GetName() const { return name\_; }

  string GetAadharNo() const { return aadhar\_no\_; }

 private:

  string name\_;

  string aadhar\_no\_;

};

class Doctor : public Person {

 public:

  Doctor(string name, string aadhar\_no, string speciality, string place\_of\_hosp)

      : Person(name, aadhar\_no), speciality\_(speciality), place\_of\_hosp\_(place\_of\_hosp) {}

  ~Doctor() {}

  string GetSpeciality() const { return speciality\_; }

  string GetPlaceOfHosp() const { return place\_of\_hosp\_; }

 private:

  string speciality\_;

  string place\_of\_hosp\_;

};

class Teacher : public Person {

 public:

  Teacher(string name, string aadhar\_no, string subject, string college\_name)

      : Person(name, aadhar\_no), subject\_(subject), college\_name\_(college\_name) {}

  ~Teacher() {}

  string GetSubject() const { return subject\_; }

  string GetCollegeName() const { return college\_name\_; }

 private:

  string subject\_;

  string college\_name\_;

};

int main() {

  Person p("John", "1234567890");

  cout << "Name: " << p.GetName() << ", Aadhar No: " << p.GetAadharNo() << endl;

  Doctor d("Jane", "0987654321", "Surgery", "St. Mary's Hospital");

  cout << "Name: " << d.GetName() << ", Aadhar No: " << d.GetAadharNo()

       << ", Speciality: " << d.GetSpeciality()

       << ", Place of Hospital: " << d.GetPlaceOfHosp() << endl;

  Teacher t("Mike", "1231231230", "Physics", "MIT");

  cout << "Name: " << t.GetName() << ", Aadhar No: " << t.GetAadharNo()

       << ", Subject: " << t.GetSubject() << ", College: " << t.GetCollegeName()

       << endl;

  return 0;

}

1. Create a class Shape with abstract methods area() and volume(). Derive classes Cube(side) and Cone(radius, height) from Shape and calculate area and volume for each shape. (20 marks)

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

  virtual double area() = 0;

  virtual double volume() = 0;

};

class Cube : public Shape {

private:

  double side;

public:

  Cube(double s) : side(s) {}

  double area() {

    return 6 \* side \* side;

  }

  double volume() {

    return side \* side \* side;

  }

};

class Cone : public Shape {

private:

  double radius;

  double height;

public:

  Cone(double r, double h) : radius(r), height(h) {}

  double area() {

    return M\_PI \* radius \* (radius + sqrt(height \* height + radius \* radius));

  }

  double volume() {

    return M\_PI \* radius \* radius \* height / 3;

  }

};

int main() {

  Cube c(2);

  Cone co(3, 4);

  cout << "Area of cube: " << c.area() << endl;

  cout << "Volume of cube: " << c.volume() << endl;

  cout << "Area of cone: " << co.area() << endl;

  cout << "Volume of cone: " << co.volume() << endl;

  return 0;

}

1. Create a class Author(name, category). Derive a class Book(Title, version, price) from A. Write print() method in both classes. Use method overriding. Show how to access the print using pointers.(20 marks)

#include <iostream>

#include <string>

using namespace std;

class Author

{

  private:

    string name;

    string category;

  public:

    Author(string name, string category)

    {

        this->name = name;

        this->category = category;

    }

    void print()

    {

        cout << "Author: " << name << " (" << category << ")" << endl;

    }

};

class Book : public Author

{

  private:

    string title;

    int version;

    double price;

  public:

    Book(string name, string category, string title, int version, double price)

        : Author(name, category)

    {

        this->title = title;

        this->version = version;

        this->price = price;

    }

    void print()

    {

        cout << "Book: " << title << " (version " << version << ", $" << price << ")" << endl;

    }

};

int main()

{

    Author \*author = new Author("Jane Smith", "Mystery");

    Book \*book = new Book("Jane Smith", "Mystery", "The Mysterious Case", 1, 19.99);

    author->print();

    book->print();

    // Deallocate the memory for the objects

    delete author;

    delete book;

    return 0;

1. Write a program to handle following file operations
   1. Read file name. Check if file exists.
   2. If file already exists read data from it and display it.
   3. Append data to the file. (20 marks)

#include <iostream>

#include <fstream>

using namespace std;

int main() {

fstream my\_file;

string myText;

my\_file.open("my\_file",ios::in);

if (!my\_file) {

cout << "File not created!";

}

else {

cout << "File created successfully!"<<endl;

//read the contents of a file

while (getline (my\_file, myText)) {

// Output the text from the file

cout << myText<<endl;

}

my\_file.close();

return 0;

}

}

10+20+5 viva=35

15+15+5 viva=35