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 |

Code :-

#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 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;

}

---------------------------------------------------------------------

//Overloading operators as friend functions

#include <iostream>

using namespace std;

class Point

{

private:

    int x;

    int y;

public:

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

    // Friend operator functions

    friend Point operator+(const Point& p1, const Point& p2)

    {

        return Point(p1.x + p2.x, p1.y + p2.y);

    }

    friend Point operator-(const Point& p1, const Point& p2)

    {

        return Point(p1.x - p2.x, p1.y - p2.y);

    }

    friend Point operator\*(const Point& p1, const Point& p2)

    {

        return Point(p1.x \* p2.x, p1.y \* p2.y);

    }

    friend Point operator/(const Point& p1, const Point& p2)

    {

        return Point(p1.x / p2.x, p1.y / p2.y);

    }

    void print()

    {

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

    }

};

int main()

{

    Point p1(1, 2);

    Point p2(3, 4);

    // friend 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. 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 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;

}

}

8 x 20 marks