DOMAIN WINTER WINNING CAMP ASSIGNMENT

Student Name: Suryansh Gehlot **UID: 22BCS10900**
Branch: BE-CSE::CS201 **Section/Group: 22BCS_FL_IOT-603/A**
Semester: 5th

➢ DAY-1 [19-12-2024]

1. Sum of Natural Numbers up to N *(Very Easy)*

Calculate the sum of all natural numbers from 1 to n, where n is a positive integer. Use the formula:
Sum=n×(n+1)/2 .
Take n as input and output the sum of natural numbers from 1 to n.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
// Function to calculate the sum of natural numbers
int sum_of_natural_numbers(int n){
    return n * (n + 1) / 2; // Using the formula
}
int main()
{
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    // Calculating the sum
    int result = sum_of_natural_numbers(n);
    // Printing the result
    cout << "The sum of all natural numbers from 1 to " << n << " is " << result << "." <<
endl;
    return 0;
}
```

Output:

2. Count Digits in a Number *(Easy)*

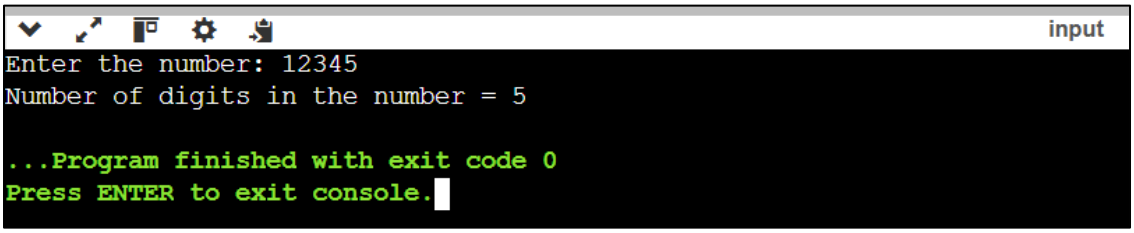
Count the total number of digits in a given number n. The number can be a positive integer. For example, for the number 12345, the count of digits is 5. For a number like 900000, the count of digits is 6.
Given an integer n, your task is to determine how many digits are present in n. This task will help you practice working with loops, number manipulation, and conditional logic.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <cmath>
int countDigits(int n) {
    if (n == 0) {
        return 1; // Special case: 0 has 1 digit
    }
    int count = 0;
    while (n != 0) {
        n /= 10; // Remove the last digit
        count++;
    }

    return count;
}
int main() {
    int n;
    cout << "Enter an integer: ";
    cin >> n;
```

```
    int digitCount = countDigits(n);
    cout << "Number of digits in " << n << " is: " << digitCount << endl;
    return 0;
}
```

Output:



3. Function Overloading for Calculating Area *(Medium)*

Write a program to calculate the area of different shapes using function overloading. Implement overloaded functions to compute the area of a circle, a rectangle, and a triangle.

Input Format:
The program should accept:
1. Radius of the circle for the first function.
2. Length and breadth of the rectangle for the second function.
3. Base and height of the triangle for the third function.

Constraints:
$1 \leq$ radius, length, breadth, base, height ≤ 103
Use 3.14159 for the value of π.

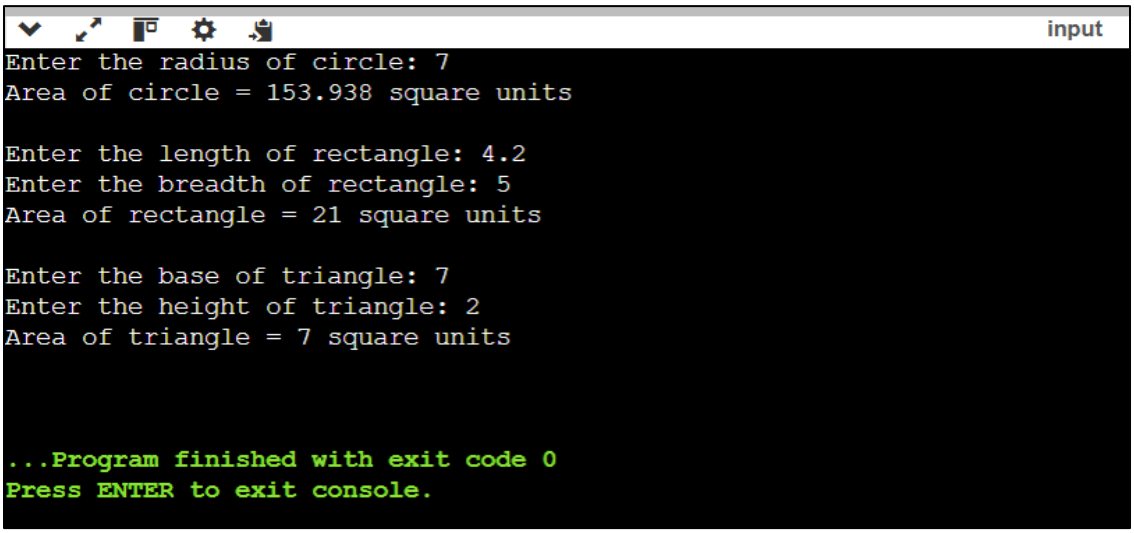
Output Format:
Print the computed area of each shape in a new line.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
```

```
#include <cmath>

const double PI = 3.14159;

// Function overloading for area calculations

double area(double radius) {
    return PI * radius * radius;
}

double area(double length, double breadth) {
    return length * breadth;
}

double area(double base, double height) {
    return 0.5 * base * height;
}

int main() {
    double radius, length, breadth, base, height;

    // Get input for circle
    cout << "Enter radius of the circle: ";
    cin >> radius;
    cout << "Area of the circle: " << area(radius) << endl;

    // Get input for rectangle
    cout << "Enter length and breadth of the rectangle: ";
    cin >> length >> breadth;
    cout << "Area of the rectangle: " << area(length, breadth) << endl;

    // Get input for triangle
    cout << "Enter base and height of the triangle: ";
    cin >> base >> height;
    cout << "Area of the triangle: " << area(base, height) << endl;

    return 0;
}
```

Output:



4. Implement Polymorphism for Banking Transactions *(Hard)*

Design a C++ program to simulate a banking system using polymorphism. Create a base class Account with a virtual method calculateInterest(). Use the derived classes SavingsAccount and CurrentAccount to implement specific interest calculation logic:

- SavingsAccount: Interest = Balance × Rate × Time.
- CurrentAccount: No interest, but includes a maintenance fee deduction.

Input Format:

1. Account Type (1 for Savings, 2 for Current).
2. Account Balance (integer).
3. For Savings Account: Interest Rate (as a percentage) and Time (in years).
4. For Current Account: Monthly Maintenance Fee.

Constraints:

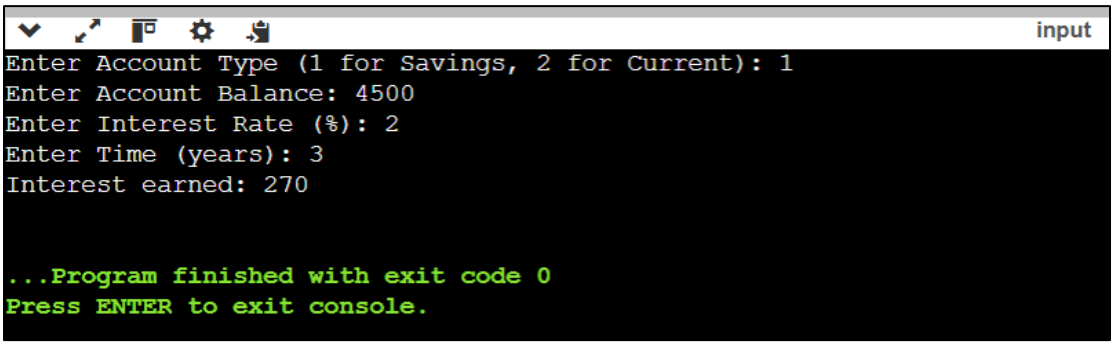
1. Account type: $1 \leq type \leq 2$.
2. Balance: $1000 \leq balance \leq 1{,}000{,}000$.
3. Interest Rate: $1 \leq rate \leq 15$.
4. Time: $1 \leq time \leq 10$.
5. Maintenance Fee: $50 \leq fee \leq 500$

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
using namespace std;

class Account {
public:
   int balance;
   Account(int b) : balance(b) {}
   virtual double calculateInterest() {
      return 0.0; // Default: No interest
   }
};

class SavingsAccount : public Account {
public:
   double rate;
   int time;

   SavingsAccount(int b, double r, int t) : Account(b), rate(r), time(t) {}

   double calculateInterest() override {
      return (balance * rate * time) / 100.0;
   }
};

class CurrentAccount : public Account {
public:
   double maintenanceFee;

   CurrentAccount(int b, double fee) : Account(b), maintenanceFee(fee) {}

   double calculateInterest() override {
      return -maintenanceFee; // Deduct maintenance fee
   }
};
```

```
int main()
{
   int accountType, balance, time;
   double rate, maintenanceFee;

   cout << "Enter Account Type (1 for Savings, 2 for Current): ";
   cin >> accountType;

   cout << "Enter Account Balance: ";
   cin >> balance;

   if (accountType == 1) {
      cout << "Enter Interest Rate (%): ";
      cin >> rate;
      cout << "Enter Time (years): ";
      cin >> time;

      SavingsAccount savingsAccount(balance, rate, time);
      cout << "Interest earned: " << savingsAccount.calculateInterest() << endl;

   }
   else if (accountType == 2)
   {
      cout << "Enter Monthly Maintenance Fee: ";
      cin >> maintenanceFee;

      CurrentAccount currentAccount(balance, maintenanceFee);
      cout << "Maintenance Fee deduction: " << currentAccount.calculateInterest() <<
endl;

   }
   else
   {
      cout << "Invalid Account Type." << endl;
   }

   return 0;
}
```

Output:



5. Hierarchical Inheritance for Employee Management System*(Very Hard)*

Create a C++ program to simulate an employee management system using hierarchical inheritance. Design a base class Employee that stores basic details (name, ID, and salary). Create two derived classes:

1. Manager: Add and calculate bonuses based on performance ratings.
2. Developer: Add and calculate overtime compensation based on extra hours worked.

The program should allow input for both types of employees and display their total earnings.

Input Format:

1.Employee Type (1 for Manager, 2 for Developer).
2.Name (string), ID (integer), and salary (integer).
3.For Manager: Performance Rating (1–5).
4.For Developer: Extra hours worked (integer).

Constraints:

- Employee type: $1 \leq \text{type} \leq 2$.
- Salary: $10{,}000 \leq \text{salary} \leq 1{,}000{,}000$.

Implementation/Code:

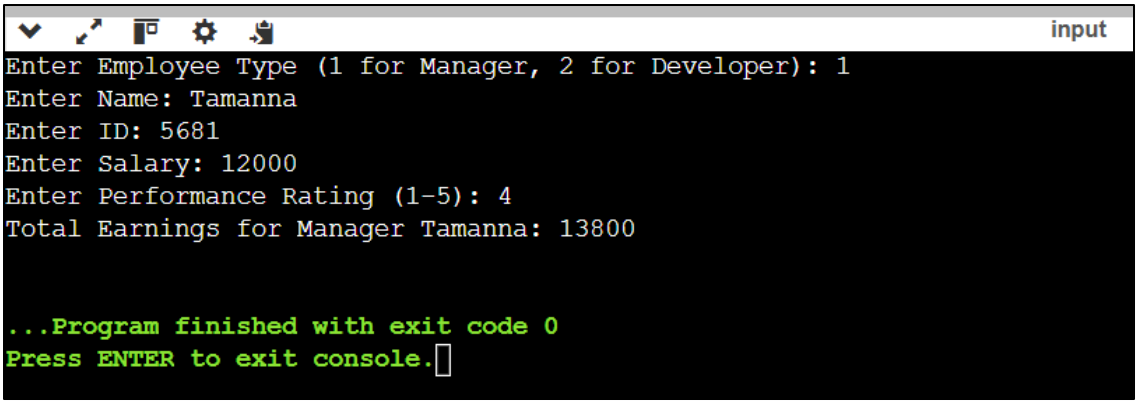
```
#include <iostream>                                    //Programming in C++
#include <string>
```

```
using namespace std;

class Employee {
public:
    string name;
    int ID;
    int salary;
    Employee(string n, int i, int s) : name(n), ID(i), salary(s) {
        if (salary < 10000 || salary > 1000000) {
            cout << "Invalid salary. Salary must be between 10000 and 1000000." << endl;
            exit(1);
        }
    }

    virtual int getTotalEarnings() {
        return salary;
    }
};

class Manager : public Employee {
public:
    int performanceRating;

    Manager(string n, int i, int s, int pr) : Employee(n, i, s), performanceRating(pr) {
        if (performanceRating < 1 || performanceRating > 5) {
            cout << "Invalid performance rating. Rating must be between 1 and 5." << endl;
            exit(1);
        }
    }

    int getTotalEarnings() override {
        int bonus = 0;
        switch (performanceRating) {
            case 1:
                bonus = 0;
                break;
            case 2:
                bonus = salary * 0.05;
```

```
            break;
        case 3:
            bonus = salary * 0.10;
            break;
        case 4:
            bonus = salary * 0.15;
            break;
        case 5:
            bonus = salary * 0.20;
            break;
    }
    return salary + bonus;
  }
};

class Developer : public Employee {
public:
  int extraHours;

  Developer(string n, int i, int s, int eh) : Employee(n, i, s), extraHours(eh) {}

  int getTotalEarnings() override {
    int overtimePay = extraHours * (salary / 200); // Assuming 200 working hours per
month
    return salary + overtimePay;
  }
};

int main() {
  int employeeType, ID, salary, performanceRating, extraHours;
  string name;

  cout << "Enter Employee Type (1 for Manager, 2 for Developer): ";
  cin >> employeeType;

  cout << "Enter Name: ";
  cin >> name;
  cout << "Enter ID: ";
```

```
    cin >> ID;
    cout << "Enter Salary: ";
    cin >> salary;

    if (employeeType == 1) {
        cout << "Enter Performance Rating (1-5): ";
        cin >> performanceRating;
        Manager manager(name, ID, salary, performanceRating);
        cout << "Total Earnings for Manager " << manager.name << ": " << manager.getTotalEarnings() << endl;
    } else if (employeeType == 2) {
        cout << "Enter Extra Hours Worked: ";
        cin >> extraHours;
        Developer developer(name, ID, salary, extraHours);
        cout << "Total Earnings for Developer " << developer.name << ": " << developer.getTotalEarnings() << endl;
    } else {
        cout << "Invalid Employee Type." << endl;
    }

    return 0;
}
```

Output:



6. Check if a Number is Prime *(Very Easy)*

Check if a given number n is a prime number. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

To determine if a number is prime, iterate from 2 to √n and check if n is divisible by any number in this range. If it is divisible, it is not a prime number; otherwise, it is a prime.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <cmath>
using namespace std;

int main()
{
   int i,num;
   cin>>num;

   if(num<2||num>100000)
   {
      cout<<"Invalid input.";
      return 1;
   }
   else
   {
      int res=sqrt(num);
      for(i=2;i<=res;i++)
      {
         if(num%i==0)
         {
            cout<<"Not Prime";
            return 1;
         }

      }
      cout<<"Prime";
   }
   return 0;
}
```

Output:

7. Print Odd Numbers up to N *(Very Easy)*

Print all odd numbers between 1 and n, inclusive. Odd numbers are integers that are not divisible by 2. These numbers should be printed in ascending order, separated by spaces. This problem is a simple introduction to loops and conditional checks. The goal is to use a loop to iterate over the numbers and check if they are odd using the condition i%2≠0.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
using namespace std;

int main()
{
   int n;
   cin>>n;

   for(int i=1;i<=n;i++)
   {
      if((i%2)!=0)
         cout<<i<<" ";
   }

   return 0;
}
```

Output:

8. Sum of Odd Numbers up to N *(Very Easy)*

Calculate the sum of all odd numbers from 1 to n. An odd number is an integer that is not divisible by 2.The sum of odd numbers, iterate through all the numbers from 1 to n, check if each number is odd, and accumulate the sum.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
using namespace std;

int main()
{
   int n,sum=0;
   cin>>n;
   if(n<1||n>10000)
   {
      cout<<"Invalid input.";
      return 1;
   }
   else
   {
      for(int i=1;i<=n;i++)
      {
         if((i%2)!=0)
            sum+=i;
      }
      cout<<sum;
   }
   return 0;
}
```

Output:

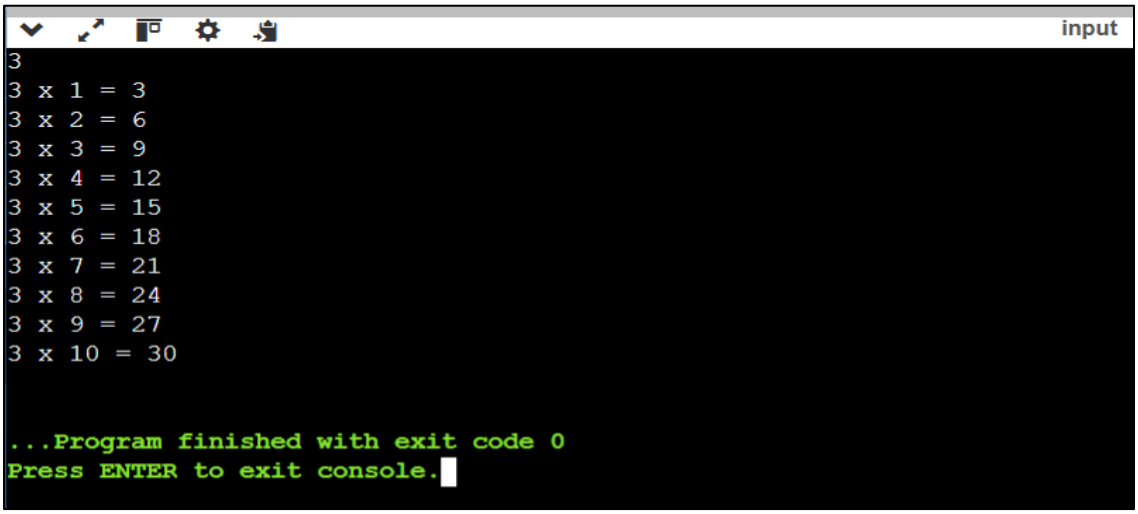
9. Print Multiplication Table of a Number *(Very Easy)*

Print the multiplication table of a given number n. A multiplication table for a number n is a list of products of n with integers from 1 to 10. For example, the multiplication table for 3 is:
3×1=3,3×2=6,…,3×10=30.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
using namespace std;
int main(){
   int n;
   cin>>n;
   if(n<1||n>100)
   {
      cout<<"Invalid input.";
      return 1;
   }
   else
   {
      for(int i=1;i<=10;i++)
      {
         cout<<n<<" x "<<i<<" = "<<n*i<<endl;
      }
   }

   return 0;
}
```

Output:



10. Reverse of a Number *(Easy)*

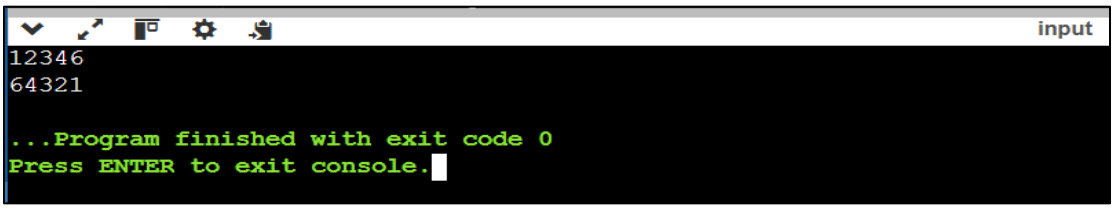
Reverse the digits of a given number n. For example, if the input number is 12345, the output should be 54321. The task involves using loops and modulus operators to extract the digits and construct the reversed number.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <cmath>
using namespace std;

int main()
{
   int n,count=0,rev=0;
   cin>>n;

   if(n<1||n>1000000000)
   {
      cout<<"Invalid input.";
      return 1;
   }
```

```
    else
    {
       int num=n;

       while(n>0)
       {
          int a=n%10;
          n=n/10;
          count++;
       }

       for(int i=count-1;i>=0;i--)
       {
          int a=num%10;
          rev=rev+(a*pow(10,i));
          num=num/10;
       }
       cout<<rev;
       return 0;
    }
}
```

Output:



11. Find the Largest Digit in a Number *(Easy)*

Find the largest digit in a given number n. For example, for the number 2734, the largest digit is 7. You need to extract each digit from the number and determine the largest one. The task will involve using loops and modulus operations to isolate the digits.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <vector>

using namespace std;

int main()
{
   int n, count = 0;
   cin >> n;
   int num = n;

   while (n > 0)
   {
      n = n / 10;
      count++;
   }

   vector<int> arr(count);
   for (int i = count - 1; i >= 0; i--)
   {
      arr[i] = num % 10;
      num = num / 10;
   }

   int largest = arr[0];
   for (int i = 1; i < count; i++)
   {
      if (arr[i] > largest)
      {
         largest = arr[i];
      }
   }

   cout << largest;
   return 0;
}
```

Output:

12. Check if a Number is a Palindrome *(Easy)*

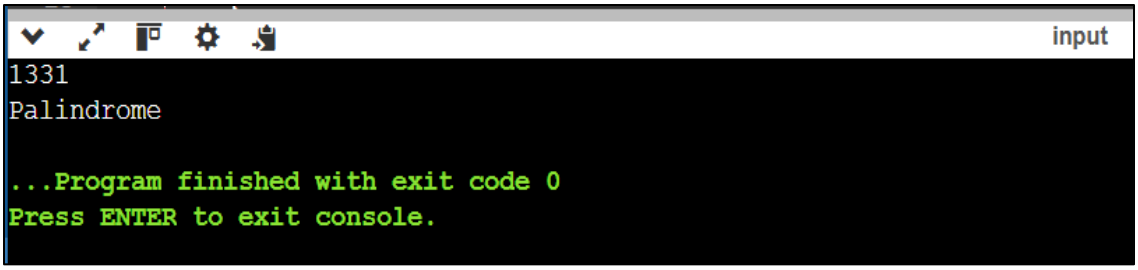
Check whether a given number is a palindrome or not. A number is called a palindrome if it reads the same backward as forward. For example, 121 is a palindrome because reading it from left to right is the same as reading it from right to left. Similarly, 12321 is also a palindrome, but 12345 is not.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <cmath>
using namespace std;

int main()
{
   int n,count=0,rev=0;
   cin>>n;

   if(n<1||n>1000000000)
   {
      cout<<"Invalid input.";
      return 1;
   }
   else
   {
      int num=n;
      int num1=n;
```

```
        while(n>0)
        {
           int a=n%10;
           n=n/10;
           count++;
        }

        for(int i=count-1;i>=0;i--)
        {
           int a=num%10;
           rev=rev+(a*pow(10,i));
           num=num/10;
        }

        if(num1==rev)
           cout<<"Palindrome";
        else
           cout<<"Not Palindrome";

        return 0;
    }
}
```

Output:



13. Find the Sum of Digits of a Number *(Easy)*

Calculate the sum of the digits of a given number n. For example, for the number 12345, the sum of the digits is 1+2+3+4+5=15. To solve this, you will need to extract each digit from the number and calculate the total sum.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <vector>
using namespace std;

int main()
{
   int n,sum=0;
   cin >> n;

   if(n<1||n>1000000000)
   {
      cout<<"Invalid input.";
      return 1;
   }
   else
   {
      while (n > 0)
      {
         int a=n%10;
         n = n / 10;
         sum+=a;
      }
      cout<<sum;
      return 0;
   }
}
```

Output:

14. Function Overloading with Hierarchical Structure *(Medium)*

Write a program that demonstrates function overloading to calculate the salary of employees at different levels in a company hierarchy. Implement overloaded functions to compute salary for:

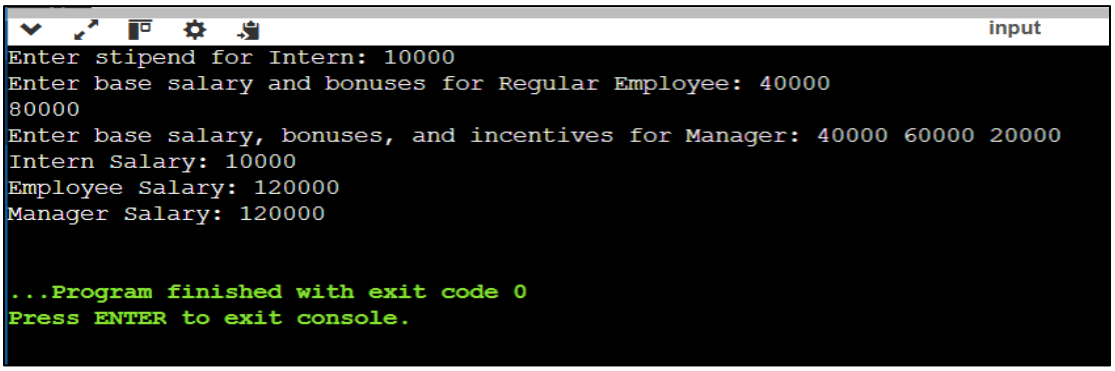
- Intern (basic stipend).
- Regular employee (base salary + bonuses).
- Manager (base salary + bonuses + performance incentives).

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
using namespace std;

// Overloaded function to calculate salary for an intern
int calculateSalary(int stipend) {
    return stipend;
}

// Overloaded function to calculate salary for a regular employee
int calculateSalary(int baseSalary, int bonuses) {
    return baseSalary + bonuses;
}

// Overloaded function to calculate salary for a manager
int calculateSalary(int baseSalary, int bonuses, int incentives) {
    return baseSalary + bonuses + incentives;
}

int main() {
    int stipend, baseSalaryEmployee, bonusesEmployee;
    int baseSalaryManager, bonusesManager, incentivesManager;

    // Input for Intern
    cout << "Enter stipend for Intern: ";
    cin >> stipend;

    // Input for Regular Employee
```

```
    cout << "Enter base salary and bonuses for Regular Employee: ";
    cin >> baseSalaryEmployee >> bonusesEmployee;

    // Input for Manager
    cout << "Enter base salary, bonuses, and incentives for Manager: ";
    cin >> baseSalaryManager >> bonusesManager >> incentivesManager;

    // Calculating and displaying salaries
    cout << "Intern Salary: " << calculateSalary(stipend) << endl;
    cout << "Employee Salary: " << calculateSalary(baseSalaryEmployee, bonusesEmployee) << endl;
    cout << "Manager Salary: " << calculateSalary(baseSalaryManager, bonusesManager, incentivesManager) << endl;

    return 0;
}
```

Output:



15. Encapsulation with Employee Details *(Medium)*

Write a program that demonstrates encapsulation by creating a class Employee. The class should have private attributes to store:

Employee ID.
Employee Name.
Employee Salary.

Provide public methods to set and get these attributes, and a method to display all details of the employee.

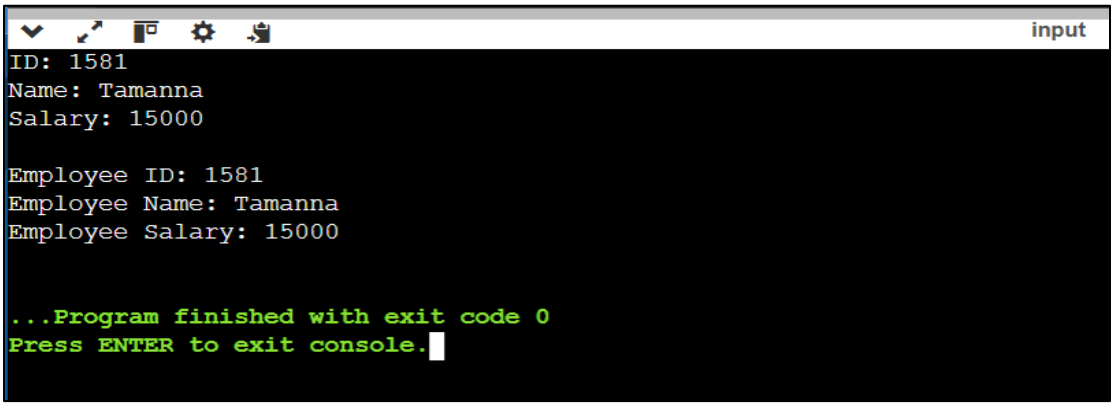
Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <string>
using namespace std;

class Employee {
private:
    int employeeID;
    string employeeName;
    float employeeSalary;

public:
    // Setter methods
    void setEmployeeID(int id) {
        if (id >= 1 && id <= 1000000) {
            employeeID = id;
        } else {
            cout << "Invalid Employee ID. Setting to default (0)." << endl;
            employeeID = 0;
        }
    }

    void setEmployeeName(string name) {
        if (name.length() <= 50) {
            employeeName = name;
        } else {
            cout << "Name is too long. Truncating to 50 characters." << endl;
            employeeName = name.substr(0, 50);
        }
    }

    void setEmployeeSalary(float salary) {
        if (salary >= 1.0 && salary <= 10000000.0) {
            employeeSalary = salary;
```

```
        } else {
            cout << "Invalid Salary. Setting to default (0.0)." << endl;
            employeeSalary = 0.0;
        }
    }

    // Getter methods
    int getEmployeeID() const {
        return employeeID;
    }

    string getEmployeeName() const {
        return employeeName;
    }

    float getEmployeeSalary() const {
        return employeeSalary;
    }

    // Method to display employee details
    void displayDetails() const {
        cout << "Employee ID: " << employeeID << endl;
        cout << "Employee Name: " << employeeName << endl;
        cout << "Employee Salary: " << employeeSalary << endl;
    }
};

int main() {
    Employee emp;

    // Input for Employee Details
    int id;
    string name;
    float salary;

    cout << "ID: ";
    cin >> id;
    cin.ignore(); // To handle newline character left in the input buffer
```

```
    cout << "Name: ";
    getline(cin, name);

    cout << "Salary: ";
    cin >> salary;
    cout<<endl;

    // Setting Employee Details
    emp.setEmployeeID(id);
    emp.setEmployeeName(name);
    emp.setEmployeeSalary(salary);

    // Displaying Employee Details
    emp.displayDetails();

    return 0;
}
```

Output:



16. Inheritance with Student and Result Classes *(Medium)*

Create a program that demonstrates inheritance by defining:

- A base class Student to store details like Roll Number and Name.
- A derived class Result to store marks for three subjects and calculate the total and percentage.

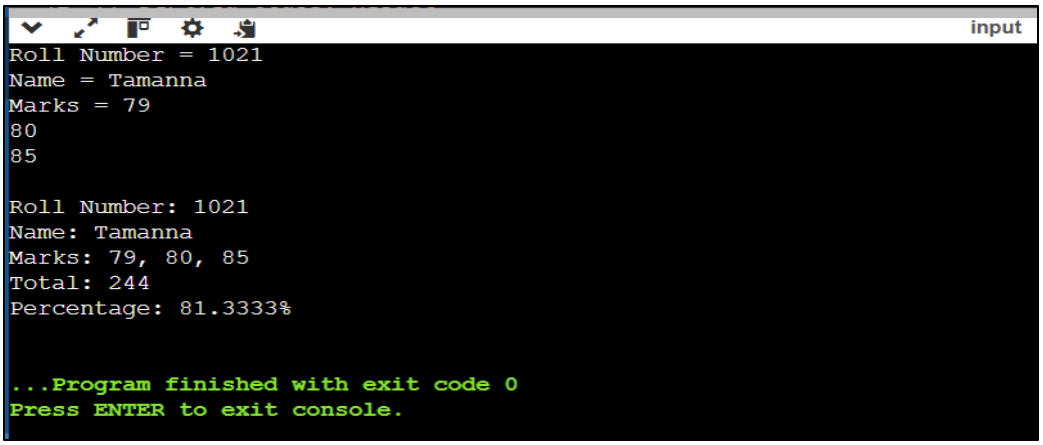
Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <string>
using namespace std;

// Base class: Student
class Student {
protected:
    int rollNumber;
    string name;

public:
    // Setter methods for Student details
    void setRollNumber(int roll) {
        if (roll >= 1 && roll <= 1000000) {
            rollNumber = roll;
        } else {
            cout << "Invalid Roll Number. Setting to default (0)." << endl;
            rollNumber = 0;
        }
    }

    void setName(string studentName) {
        if (studentName.length() <= 50) {
            name = studentName;
        } else {
            cout << "Name is too long. Truncating to 50 characters." << endl;
            name = studentName.substr(0, 50);
        }
    }

    // Getter methods for Student details
    int getRollNumber() const {
        return rollNumber;
    }
```

```
    string getName() const {
        return name;
    }
};

// Derived class: Result
class Result : public Student {
private:
    int marks[3]; // Array to store marks for three subjects

public:
    // Setter method for marks
    void setMarks(int m1, int m2, int m3) {
        if (m1 >= 0 && m1 <= 100 && m2 >= 0 && m2 <= 100 && m3 >= 0 && m3 <=
100) {
            marks[0] = m1;
            marks[1] = m2;
            marks[2] = m3;
        } else {
            cout << "Invalid marks entered. Setting all marks to 0." << endl;
            marks[0] = marks[1] = marks[2] = 0;
        }
    }

    // Method to calculate total marks
    int calculateTotal() const {
        return marks[0] + marks[1] + marks[2];
    }

    // Method to calculate percentage
    float calculatePercentage() const {
        return (calculateTotal() / 300.0f) * 100;
    }

    // Method to display student details and result
    void displayDetails() const {
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Name: " << name << endl;
```

```
        cout << "Marks: " << marks[0] << ", " << marks[1] << ", " << marks[2] << endl;
        cout << "Total: " << calculateTotal() << endl;
        cout << "Percentage: " << calculatePercentage() << "%" << endl;
    }
};

int main() {
    Result student;

    // Input for student details
    int roll;
    string name;
    int m1, m2, m3;

    cout << "Roll Number = ";
    cin >> roll;
    cin.ignore(); // To handle newline character left in the input buffer

    cout << "Name = ";
    getline(cin, name);

    cout << "Marks = ";
    cin >> m1 >> m2 >> m3;
    cout<<endl;

    // Setting student details and marks
    student.setRollNumber(roll);
    student.setName(name);
    student.setMarks(m1, m2, m3);

    // Displaying student details and result
    student.displayDetails();

    return 0;
}
```

Output:



17. Polymorphism with Shape Area Calculation *(Medium)*

Create a program that demonstrates polymorphism by calculating the area of different shapes using a base class Shape and derived classes for Circle, Rectangle, and Triangle. Each derived class should override a virtual function to compute the area of the respective shape.

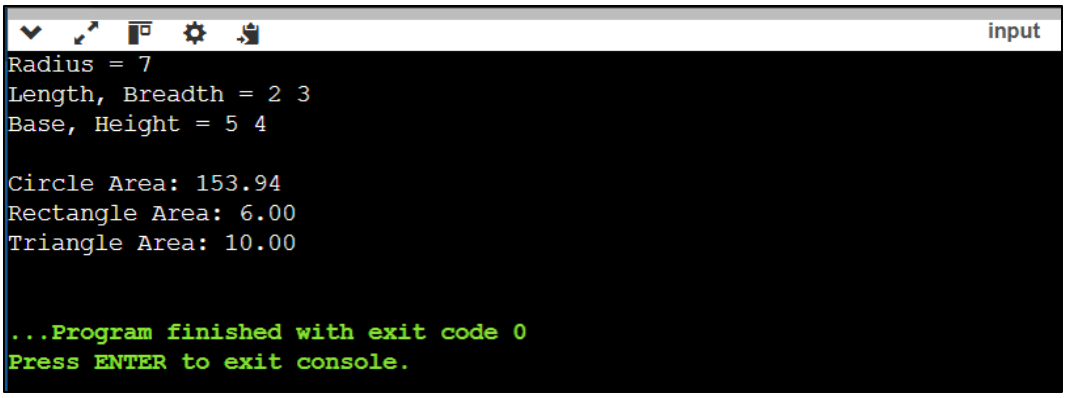
Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <iomanip> // For setting precision
#include <cmath>   // For mathematical operations
using namespace std;

// Base class: Shape
class Shape {
public:
    virtual double calculateArea() const = 0; // Pure virtual function
    virtual ~Shape() {} // Virtual destructor
};

// Derived class: Circle
class Circle : public Shape {
private:
    double radius;
```

```
public:
    Circle(double r) : radius(r) {}

    double calculateArea() const override {
        const double pi = 3.14159;
        return pi * radius * radius;
    }
};

// Derived class: Rectangle
class Rectangle : public Shape {
private:
    double length, breadth;

public:
    Rectangle(double l, double b) : length(l), breadth(b) {}

    double calculateArea() const override {
        return length * breadth;
    }
};

// Derived class: Triangle
class Triangle : public Shape {
private:
    double base, height;

public:
    Triangle(double b, double h) : base(b), height(h) {}

    double calculateArea() const override {
        return 0.5 * base * height;
    }
};

int main()
{
    // Input for Circle
```

```
    double radius;
    cout << "Radius = ";
    cin >> radius;

    // Input for Rectangle
    double length, breadth;
    cout << "Length, Breadth = ";
    cin >> length >> breadth;

    // Input for Triangle
    double base, height;
    cout << "Base, Height = ";
    cin >> base >> height;
    cout<<endl;

    // Creating objects dynamically
    Shape* circle = new Circle(radius);
    Shape* rectangle = new Rectangle(length, breadth);
    Shape* triangle = new Triangle(base, height);

    // Displaying areas with polymorphism
    cout << fixed << setprecision(2);
    cout << "Circle Area: " << circle->calculateArea() << endl;
    cout << "Rectangle Area: " << rectangle->calculateArea() << endl;
    cout << "Triangle Area: " << triangle->calculateArea() << endl;

    // Cleaning up memory

    delete circle;
    delete rectangle;
    delete triangle;

    return 0;
}
```

Output:



18. Implementing Polymorphism for Shape Hierarchies *(Hard)*

Write a program to demonstrate runtime polymorphism in C++ using a base class Shape and derived classes Circle, Rectangle, and Triangle. The program should use virtual functions to calculate and print the area of each shape based on user input.

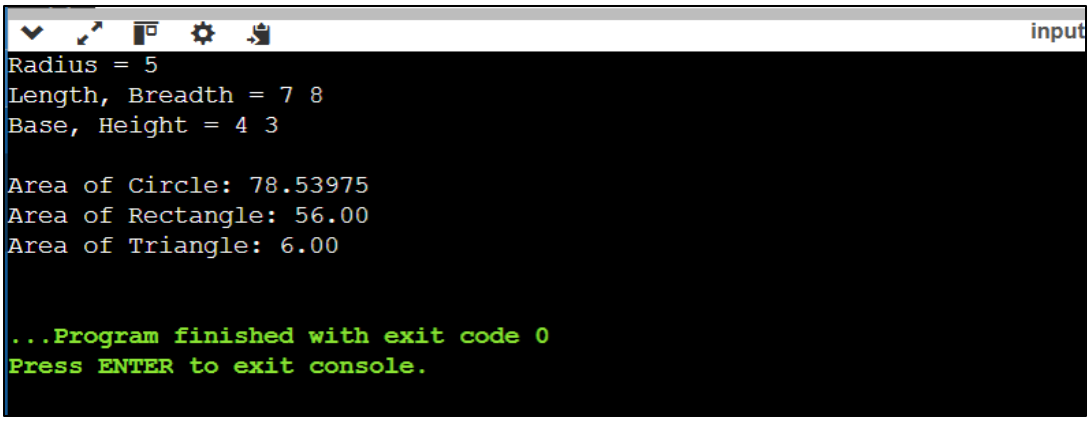
Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <iomanip> // For setting precision
#include <cmath>   // For mathematical operations
using namespace std;

// Base class: Shape
class Shape {
public:
    virtual double calculateArea() const = 0; // Pure virtual function
    virtual ~Shape() {} // Virtual destructor
};

// Derived class: Circle
class Circle : public Shape {
private:
    double radius;

public:
    Circle(double r) : radius(r) {}
```

```
    double calculateArea() const override {
        const double pi = 3.14159;
        return pi * radius * radius;
    }
};

// Derived class: Rectangle
class Rectangle : public Shape {
private:
    double length, breadth;

public:
    Rectangle(double l, double b) : length(l), breadth(b) {}

    double calculateArea() const override {
        return length * breadth;
    }
};

// Derived class: Triangle
class Triangle : public Shape {
private:
    double base, height;

public:
    Triangle(double b, double h) : base(b), height(h) {}

    double calculateArea() const override {
        return 0.5 * base * height;
    }
};

int main()
{
    // Input for Circle
    double radius;
    cout << "Radius = ";
```

```
    cin >> radius;

    // Input for Rectangle
    double length, breadth;
    cout << "Length, Breadth = ";
    cin >> length >> breadth;

    // Input for Triangle
    double base, height;
    cout << "Base, Height = ";
    cin >> base >> height;
    cout<<endl;

    // Creating objects dynamically

    Shape* circle = new Circle(radius);
    Shape* rectangle = new Rectangle(length, breadth);
    Shape* triangle = new Triangle(base, height);

    // Displaying areas with polymorphism

    cout << fixed << setprecision(5);
    cout << "Area of Circle: " << circle->calculateArea() << endl;
    cout << "Area of Rectangle: " << fixed << setprecision(2) << rectangle-
>calculateArea() << endl;
    cout << "Area of Triangle: " << triangle->calculateArea() << endl;

    // Cleaning up memory

    delete circle;
    delete rectangle;
    delete triangle;

    return 0;
}
```

Output:



19. Advanced Function Overloading for Geometric Shapes *(Very Hard)*

Create a C++ program that demonstrates **function overloading** to calculate the area of different geometric shapes. Implement three overloaded functions named calculateArea that compute the area for the following shapes:

Circle: Accepts the radius.

Rectangle: Accepts the length and breadth.

Triangle: Accepts the base and height.

Additionally, use a menu-driven program to let the user choose the type of shape and input the respective parameters. Perform necessary validations on the input values.

Implementation/Code:

```
#include <iostream>                                    //Programming in C++
#include <iomanip> // For setting precision
using namespace std;

// Function to calculate area of a circle
double calculateArea(double radius) {
    const double pi = 3.14159;
    return pi * radius * radius;
}

// Function to calculate area of a rectangle
double calculateArea(double length, double breadth) {
```

```
    return length * breadth;
}

// Function to calculate area of a triangle
float calculateArea(float base, float height) {
    return 0.5 * base * height;
}

int main() {
    int choice;
    cout << "Select the shape to calculate area:" << endl;
    cout << "1. Circle" << endl;
    cout << "2. Rectangle" << endl;
    cout << "3. Triangle" << endl;
    cout << "Enter your choice (1-3): ";
    cin >> choice;

    if (choice == 1) {
        // Circle
        double radius;
        cout << "Enter the radius of the circle: ";
        cin >> radius;
        if (radius <= 0) {
            cout << "Invalid input. Radius must be positive." << endl;
        } else {
            cout << fixed << setprecision(3);
            cout << "Shape: Circle" << endl;
            cout << "Radius: " << radius << endl;
            cout << "Area: " << calculateArea(radius) << endl;
        }
    } else if (choice == 2) {
        // Rectangle
        double length, breadth;
        cout << "Enter the length and breadth of the rectangle: ";
        cin >> length >> breadth;
        if (length <= 0 || breadth <= 0) {
            cout << "Invalid input. Length and breadth must be positive." << endl;
        } else {
```

```
            cout << fixed << setprecision(3);
            cout << "Shape: Rectangle" << endl;
            cout << "Length: " << length << endl;
            cout << "Breadth: " << breadth << endl;
            cout << "Area: " << calculateArea(length, breadth) << endl;
        }
    } else if (choice == 3) {
        // Triangle
        float base, height;
        cout << "Enter the base and height of the triangle: ";
        cin >> base >> height;
        if (base <= 0 || height <= 0) {
            cout << "Invalid input. Base and height must be positive." << endl;
        } else {
            cout << fixed << setprecision(3);
            cout << "Shape: Triangle" << endl;
            cout << "Base: " << base << endl;
            cout << "Height: " << height << endl;
            cout << "Area: " << calculateArea(base, height) << endl;
        }
    } else {
        // Invalid choice
        cout << "Invalid choice. Please select a valid shape type." << endl;
    }
    return 0;
}
```

Output: