

DOMAIN WINTER WINNING CAMP

Student Name: Nainsi

UID: 22BCS15333

Branch: BE CSE

Section/Group: 620 A

DAY 1:

QUES 1: Sum of Natural Numbers up to N

Calculate the sum of all natural numbers from 1 to n, where n is a positive integer. Use the formula: $\text{Sum} = n \times (n+1) / 2$.

Take n as input and output the sum of natural numbers from 1 to n.

Solution:

```
#include <iostream>

using namespace std;

int main() {
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    if (n > 0) {
        int sum = n * (n + 1) / 2;
        cout << "The sum of natural numbers from 1 to " << n << " is: " << sum << endl;
    } else {
        cout << "Please enter a positive integer." << endl;
    }
    return 0;
}
```

```
Enter a positive integer: 32
The sum of natural numbers from 1 to 32 is: 528
```

QUES 2: Check if a Number is Prime

Objective 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 \sqrt{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.

Solution:

```
#include <iostream>

#include <cmath>

using namespace std;

bool isPrime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i <= sqrt(n); i++)
        { if (n % i == 0) {
            return false;
        }
    }
    return true;
}

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
    if (n >= 2 && n <= 100000)
        { if (isPrime(n)) {
            cout << "Prime" << endl;
        } else {
            cout << "Not Prime" << endl;
        }
    } else {
        cout << "Number out of range. Please enter a number between 2 and 100000." << endl;
```

```
}  
return 0;  
}
```

```
Enter a number: 7  
Prime
```

QUES 3: Print Multiplication Table of a Number

Objective: 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:

$3 \times 1 = 3, 3 \times 2 = 6, \dots, 3 \times 10 = 30$.

Solution:

```
#include <iostream>  
  
using namespace  
std; int main() {  
    int n;  
    cout << "Enter a number: ";  
    cin >> n;  
    for (int i = 1; i <= 10; i++) {  
        cout << n << " x " << i << " = " << n * i << endl;  
    }  
}
```

```
Enter a number: 7  
7 x 1 = 7  
7 x 2 = 14  
7 x 3 = 21  
7 x 4 = 28  
7 x 5 = 35  
7 x 6 = 42  
7 x 7 = 49  
7 x 8 = 56  
7 x 9 = 63  
7 x 10 = 70
```

```
return 0;}
```

QUES 4: Sum of Odd Numbers up to N

Objective: 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.

Solution:

```
#include <iostream>

using namespace

std; int main() {
    int n, sum = 0;
    cout << "Enter a number: ";

    cin >> n;

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

    cout << "Sum of odd numbers up to " << n << " is: " << sum << endl;

    return 0;
}
```

```
Enter a number: 5
Sum of odd numbers up to 5 is: 9
```

QUES 5: Count Digits in a Number

Objective: 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.

Solution:

```
#include <iostream>

using namespace
std; int main() {
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    if (n > 0) {
        int count = 0;
        while (n > 0) {
            n /= 10;
            count++;
        }
        cout << "The number of digits is: " << count << endl;
    } else {
        cout << "Please enter a positive integer." << endl;
    }
    return 0;
}
```

```
Enter a positive integer: 67353428
The number of digits is: 8
```

QUES 6: Check if a Number is a Palindrome

Objective: 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.

Solution:

```
#include <iostream>

using namespace

std; int main() {

    int n, originalNumber, reversedNumber = 0;

    cout << "Enter a number: ";

    cin >> n;

    originalNumber = n;

    while (n > 0) {

        int digit = n % 10;

        reversedNumber = reversedNumber * 10 + digit;

        n /= 10;

    }

    if (originalNumber == reversedNumber) {

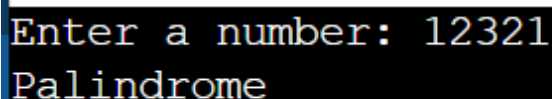
        cout << "Palindrome" << endl;

    } else {

        cout << "Not a Palindrome" << endl;    }

    return 0;

}
```



```
Enter a number: 12321
Palindrome
```

QUES 7: Function Overloading for Calculating Area.

Objective: 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.

Solution:

```
#include <iostream>

using namespace std;

double calculateArea(double radius) {
    return 3.14159 * radius * radius;
}

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

double calculateArea(double base, double height, int isTriangle) {
    return 0.5 * base * height;
}

int main() {
    double radius, length, breadth, base,
    height; cout << "Enter the radius of the
    circle: "; cin >> radius;

    cout << "Area of the circle: " << calculateArea(radius) <<
    endl; cout << "Enter the length and breadth of the rectangle: ";
    cin >> length >> breadth;

    cout << "Area of the rectangle: " << calculateArea(length, breadth) << endl;
    cout << "Enter the base and height of the triangle: ";
    cin >> base >> height;

    cout << "Area of the triangle: " << calculateArea(base, height, 1) << endl;
    return 0;
}
```



```
Enter the radius of the circle: 6
Area of the circle: 113.097
Enter the length and breadth of the rectangle: 7
8
Area of the rectangle: 56
Enter the base and height of the triangle: 9
10
Area of the triangle: 45
```


QUES 8: Encapsulation with Employee Details

Objective: 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.

Solution:

```
#include <iostream>

#include <string>

using namespace std;

class Employee {
private:
    int employeeID;
    string employeeName;
    double
    employeeSalary;
public:
    void setEmployeeID(int id) {
        employeeID = id;
    }
    void setEmployeeName(string name) {
        employeeName = name;
    }
    void setEmployeeSalary(double salary) {
        employeeSalary = salary;
    }
    int getEmployeeID() const {
        return employeeID;
    }
}
```

```
}  
  
string getEmployeeName() const {  
    return employeeName;  
}  
  
double getEmployeeSalary() const {  
    return employeeSalary;  
}  
  
void displayDetails() const {  
    cout << "Employee ID: " << employeeID << endl;  
    cout << "Employee Name: " << employeeName << endl;  
    cout << "Employee Salary: $" << employeeSalary << endl;  
}  
};  
  
int main() {  
    Employee emp;  
    emp.setEmployeeID(101);  
    emp.setEmployeeName("John Doe");  
    emp.setEmployeeSalary(55000.50);  
    cout << "Employee Details:" <<  
    endl; emp.displayDetails();  
    return 0;  
}
```

```
Employee Details:  
Employee ID: 101  
Employee Name: John Doe  
Employee Salary: $55000.5
```

QUES 9: Inheritance with Student and Result Classes.

Objective: 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.

Solution:

```
#include <iostream>
#include <string>
using namespace std;
class Student {
protected:
    int
    rollNumber;
    string name;
public:
    void setDetails(int r, string n) {
        rollNumber = r;
        name = n;
    }
    void displayDetails() const {
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Name: " << name << endl;
    }
};
class Result : public Student {
private:
    float marks[3];
public:
    void setMarks(float m1, float m2, float m3) {
        marks[0] = m1;
        marks[1] = m2;
        marks[2] = m3;
    }
    float calculateTotal() const {
```

```
        return marks[0] + marks[1] + marks[2];
    }

    float calculatePercentage() const {
        return (calculateTotal() / 300) * 100; // Assuming each subject is out of 100
    }

    void displayResult() const {
        displayDetails(); // Call base class
        method

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

int main() {
    Result student;
    student.setDetails(101,
        "Alice"); student.setMarks(85,
        90, 88);
    cout << "Student Result Details:" << endl;
    student.displayResult();
    return 0;
}
```

```
Student Result Details:
Area of Rectangle: 15
Area of Circle: 153.938
Area of Triangle: 12
Total Marks: 263
Percentage: 87.6667%
```

QUES 10: Implement Polymorphism for Banking Transactions

Objective: 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: $\text{Interest} = \text{Balance} \times \text{Rate} \times \text{Time}$.
- CurrentAccount: No interest, but includes a maintenance fee deduction.

Solution:

```
#include <iostream>

using namespace std;

class Account {
protected:
    double balance;
public:
    Account(double bal) : balance(bal) {}
    virtual void calculateInterest() = 0;
```

```
void displayBalance() const {
    cout << "Balance: $" << balance << endl;
}

virtual ~Account() {}
};

class SavingsAccount : public Account {
private:
    double
    interestRate;
    double time;
public:
    SavingsAccount(double bal, double rate, double t) : Account(bal), interestRate(rate),
    time(t) {}

    void calculateInterest() override {
        double interest = balance * (interestRate / 100) * time;
        balance += interest;
        cout << "Interest calculated for Savings Account: $" << interest << endl;
    }
};

class CurrentAccount : public Account
{ private:
    double
    maintenanceFee; public:
    CurrentAccount(double bal, double fee) : Account(bal), maintenanceFee(fee) {}

    void calculateInterest() override {
        balance -= maintenanceFee;

        cout << "Maintenance fee deducted for Current Account: $" << maintenanceFee <<
endl;
    }
};

int main() {
```

```
Account* savingsAcc = new SavingsAccount(10000, 5, 2);
savingsAcc->displayBalance();
savingsAcc->calculateInterest()
;
savingsAcc->displayBalance();
cout << endl;

Account* currentAcc = new CurrentAccount(5000,
50); currentAcc->displayBalance();
currentAcc->calculateInterest()
;
currentAcc->displayBalance();
delete savingsAcc;
delete currentAcc;
return 0;
}
```

```
Balance: $10000
Interest calculated for Savings Account: $1000
Balance: $11000

Balance: $5000
Maintenance fee deducted for Current Account: $50
Balance: $4950
```

QUES 11: Implement Polymorphism for Banking Transactions

Objective: 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: $\text{Interest} = \text{Balance} \times \text{Rate} \times \text{Time}$.

CurrentAccount: No interest, but includes a maintenance fee deduction.

Solution:

```
#include <iostream>

using namespace std;

class Account {
protected;
```

```
cout << "Enter Account Type (1 for Savings, 2 for Current):  
"; cin >> accountType;  
if (accountType == 1)  
{ double balance,  
rate; int time;  
cout << "Enter Balance: ";  
cin >> balance;  
cout << "Enter Interest Rate (%):  
"; cin >> rate;  
cout << "Enter Time (in years): ";  
cin >> time;  
if (balance >= 1000 && rate >= 1 && rate <= 15 && time >= 1 && time <= 10) {  
    SavingsAccount sa(balance, rate, time);  
    sa.calculateInterest();  
} else {  
    cout << "Invalid input for Savings Account." << endl;  
}  
} else if (accountType == 2) {  
    double balance, fee;  
    cout << "Enter Balance: ";  
    cin >> balance;  
    cout << "Enter Monthly Maintenance Fee: ";  
    cin >> fee;  
    if (balance >= 1000 && fee >= 50 && fee <= 500)  
    { CurrentAccount ca(balance, fee);  
    ca.calculateInterest();  
} else {  
    cout << "Invalid input for Current Account." << endl;  
}  
}
```



```
cout << "Enter Account Type (1 for Savings, 2 for Current):  
"; cin >> accountType;  
if (accountType == 1)  
{ double balance,  
rate; int time;  
cout << "Enter Balance: ";  
cin >> balance;  
cout << "Enter Interest Rate (%):  
"; cin >> rate;  
cout << "Enter Time (in years): ";  
cin >> time;  
if (balance >= 1000 && rate >= 1 && rate <= 15 && time >= 1 && time <= 10) {  
    SavingsAccount sa(balance, rate, time);  
    sa.calculateInterest();  
} else {  
    cout << "Invalid input for Savings Account." << endl;  
}  
} else if (accountType == 2) {  
    double balance, fee;  
    cout << "Enter Balance: ";  
    cin >> balance;  
    cout << "Enter Monthly Maintenance Fee: ";  
    cin >> fee;  
    if (balance >= 1000 && fee >= 50 && fee <= 500)  
    { CurrentAccount ca(balance, fee);  
    ca.calculateInterest();  
} else {  
    cout << "Invalid input for Current Account." << endl;  
}  
}
```

```
cout << "Enter Account Type (1 for Savings, 2 for Current):  
"; cin >> accountType;  
if (accountType == 1)  
{ double balance,  
rate; int time;  
cout << "Enter Balance: ";  
cin >> balance;  
cout << "Enter Interest Rate (%):  
"; cin >> rate;  
cout << "Enter Time (in years): ";  
cin >> time;  
if (balance >= 1000 && rate >= 1 && rate <= 15 && time >= 1 && time <= 10) {  
    SavingsAccount sa(balance, rate, time);  
    sa.calculateInterest();  
} else {  
    cout << "Invalid input for Savings Account." << endl;  
}  
} else if (accountType == 2) {  
    double balance, fee;  
    cout << "Enter Balance: ";  
    cin >> balance;  
    cout << "Enter Monthly Maintenance Fee: ";  
    cin >> fee;  
    if (balance >= 1000 && fee >= 50 && fee <= 500)  
    { CurrentAccount ca(balance, fee);  
    ca.calculateInterest();  
} else {  
    cout << "Invalid input for Current Account." << endl;  
}  
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
} else {  
    cout << "Invalid Account Type. Please enter 1 or 2." << endl;  
}  
return 0;  
}
```

```
Enter Account Type (1 for Savings, 2 for Current): 1  
Enter Balance: 45000  
Enter Interest Rate (%): 5  
Enter Time (in years): 6  
Savings Account Interest: 13500
```