Name : Arin Rai
UID : 22BCS11773
Class : KPIT - 901

Q1. Calculate the sum of all natural numbers from 1 to n.
Q2. Count the total number of digits in a given number n.
Q3. 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.
Q4. Design a program to simulate a banking system using polymorphism. Create a base class Account with a virtual method calculateInterest().Use the derived classes to implement specific interest calculation logic:

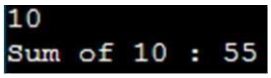
- SavingsAccount : Interest = Balance × Rate × Time.
- CurrentAccount : No interest, but includes a maintenance fee deduction. Q5. Create a C++ program to simulate an employee management system using hierarchical inheritance. Design a base class Employee that stores basic details (name, ID, and salary). Create two derived classes:
- Manager: Add and calculate bonuses based on performance ratings.
- Developer: Add and calculate overtime compensation based on extra hours worked. The program should allow input for both types of employees and display them total earnings.

Solutions :

A1. Sum of natural numbers upto n.

```
#include <iostream>

int sum(int n) { return n * (n + 1) / 2; }
 int main() {     int n = 0;
std::cin >> n;
    std::cout << "Sum of " << n << " : " << sum(n); }
```

Output :



A2. Count digits in a number.

```
#include <iostream>

int count_digits(int n) {     int
count = 0;     while(n) {
count++;         n /= 10;
   }
   return count;
}

int main() {     int n = 0;
std::cin >> n;
   std::cout << "Digits in " << n << " : " << count_digits(n); }
```

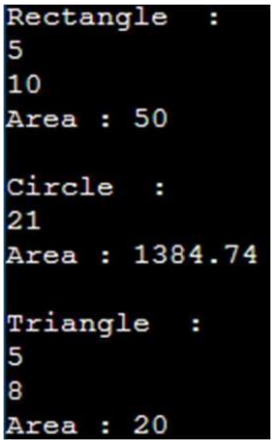
Output :

```
1056781
Digits in 1056781 : 7
```

A3. Function Overloading for Calculating Area.

```
#include <iostream>

int area(int l, int b) { return l * b; }

float area(int r) { return 3.14 * r * r; }

float area(float h, float b) { return 0.5 * b * h; }

int main() {
   int r_l, r_b, c_r = 0;     float t_b, t_h = 0;

   std::cout << "Rectangle  : \n";     std::cin >> r_l;
std::cin >> r_b;
   std::cout << "Area : " << area(r_l, r_b) << "\n";     std::cout << "\nCircle
: \n";     std::cin >> c_r;
   std::cout << "Area : " << area(c_r) << "\n";     std::cout <<
"\nTriangle   : \n";       std::cin >> t_b;       std::cin >> t_h;
std::cout << "Area : " << area(t_b, t_h) << "\n"; }
```

Output :



A4. Implement Polymorphism for Banking Transactions

```
#include <iostream>

using namespace std;

typedef unsigned int uint;
class Account {     protected:
uint Acc_Id;         uint
Acc_Balance;
public:
       Account(uint Acc_Id, uint Acc_Balance) {            this->Acc_Id      =
Acc_Id;            this->Acc_Balance = Acc_Balance;
       }

       virtual ~Account() = default;

       virtual void calculate_interest() = 0;  };  class
Savings_Account : public Account {
private:          float rate;         uint  time;       public:
       Savings_Account(             uint
Acc_Id,              uint Acc_Balance,
float rate,              uint time
       ) : Account(Acc_Id, Acc_Balance) {            this>rate
= rate;            this->time = time;
       }

       void calculate_interest() {
          float interest = this->rate * this->time * this->Acc_Balance;
          cout << "\nCurrent Balance : " << this->Acc_Balance + interest << "\n";        }
};  class Current_Account : public Account {
private:
       uint monthly_maintenance;         uint
months_elapsed;
public:
       Current_Account(            uint Acc_Id,              uint
Acc_Balance,                  uint   monthly_maintenance,
uint  months_elapsed
```

```
    ) : Account(Acc_Id, Acc_Balance) {          this->monthly_maintenance = monthly_maintenance;
this>months_elapsed    = months_elapsed;
    }

    void calculate_interest() {
        float maintenance_charges = months_elapsed * monthly_maintenance;
        if (this->Acc_Balance != 0 && this->Acc_Balance > maintenance_charges)
        {
            cout << "\nCurrent Balance : "
                << this->Acc_Balance  - maintenance_charges << "\n";          }
    }
};
int main(int argc, char* argv[]) {    int n = 0;
cin >> n;     switch (n) {
case 1: {
        uint  Balance = 0;          float Rate
= 0.0;          uint  Time    = 0;          cout
<< "Balance : ";          cin >> Balance;          cout <<
"Rate : ";
cin >> Rate;          cout << "Time : ";          cin
>> Time;

        Savings_Account acc(
            1,
            Balance,
            Rate,
            Time
        );

        acc.calculate_interest();

        break;
    }

    case 2: {
        uint Balance          = 0;          uint
Monthly_Maintenance = 0;          uint Months_Elapsed    =
0;

        cout << "Balance : ";          cin >> Balance;
cout << "Monthly Maintenance : ";          cin >>
Monthly_Maintenance;          cout << "Months Elapsed : ";          cin
>> Months_Elapsed;


        Current_Account acc(
            1,
            Balance,
            Monthly_Maintenance,
            Months_Elapsed
        );

        acc.calculate_interest();
```

```
            break;
        }
        default:            cout << "Invalid Choice !";
    }       return 0;  }
```

Output :

A5. Hierarchical Inheritance for Employee Management System

```
#include <iostream>
#include <limits>

#define BONUS_FOR_RATING 0.1
#define OVERTIME_RATE    500

using    namespace  std;  typedef  unsigned int
uint;

class Employee {     protected:        uint
Emp_Id;        string
Emp_Name;        uint
Emp_Salary;
```

```
    public:
        Employee(uint Emp_Id, string& Emp_Name, uint Emp_Salary) {
    this>Emp_Id     = Emp_Id;           this->Emp_Name   = Emp_Name;
    this>Emp_Salary = Emp_Salary;
        }

        virtual void Display_Employee() = 0;
    };
    class Manager : public Employee {
    private:        uint  Rating;        float Bonus;
    public:
        Manager(uint Emp_Id, string& Emp_Name, uint Emp_Salary, uint Rating)
```

```
        : Employee(Emp_Id, Emp_Name, Emp_Salary) {          if (Rating < MIN_RATING || Rating >
  MAX_RATING) { this->Rating = 1; }          else                                { this->Rating = Rating; }
                        this>Bonus  = this->Emp_Salary * this->Rating * BONUS_FOR_RATING;       }

    void Display_Employee() {
      cout << "Name   : " << this->Emp_Name            << endl            << "ID    : " << this>Emp_Id              <<
endl
          << "Salary : " << this->Emp_Salary          << endl
          << "Bonus  : " << this->Bonus               << endl
          << "Total  : " << this->Emp_Salary + this->Bonus << endl;
    }
};
class Developer : public Employee {
private:      uint Overtime_Hours;
uint Overtime_Compensation;          public:
    Developer(uint Emp_Id, string& Emp_Name, uint Emp_Salary, uint Overtime_Hours) :       Employee(Emp_Id,
Emp_Name, Emp_Salary) {          this->Overtime_Hours      = Overtime_Hours;
      this->Overtime_Compensation = this->Overtime_Compensation * OVERTIME_RATE;       }

    void Display_Employee() {
      cout << "Name              : " << this->Emp_Name << endl            << "ID              : " << this->Emp_Id
<< endl
          << "Salary            : " << this->Emp_Salary << endl
          << "Overtime Compensation : " << this->Overtime_Compensation << endl
          << "Total             : " << this->Emp_Salary +
                                       this->Overtime_Compensation << endl;      }
};
int main(int argc, char* argv[]) {    int  n    =
0;    uint id    = 0;    uint salary = 0;    uint
rating = 0;    uint ot_hrs = 0;    string Name;
```

```
    cin >> n;
    cin.ignore(numeric_limits<streamsize>::max(), '\n');

    switch (n) {        case 1: {
cout << "Name : ";            getline(cin,
Name);            cin.clear();
cin.sync();            cout << "Salary : ";
cin >> salary;
            cout << "Rating (1 - 5) : ";            cin >> rating;

            Manager mgr(1, Name, salary, rating);
mgr.Display_Employee();            break;
        }

        case 2: {
            cout      <<      "Name      :      ";
getline(cin, Name);            cin.clear();
cin.sync();            cout << "Salary : ";
cin >> salary;
            cout << "Overtime Hours : ";            cin >> ot_hrs;

            Developer dev(1, Name, salary, ot_hrs);
dev.Display_Employee();            break;
        }    }
    return 0;
}
```

Output :