**Q. Phone Book Program**

class PhoneBook{

public static void Main(String[] args) {

String[] names = new String[10]; // at max Ten Contacts

int[] numbers = new int[10]; // for corresponding 10 phone numbers

int total = 0;

while (true) {

Console.WriteLine("Make Choice : ");

Console.WriteLine("1)Add Contact");

Console.WriteLine("2)Search Contact");

Console.WriteLine("3)Edit Contact");

Console.WriteLine("4)Delete Contact");

Console.WriteLine("0)Exit");

int choice = int.Parse(Console.ReadLine()!);

Console.WriteLine("Your Choice : " + choice);

switch (choice) {

case 0:

Console.WriteLine("Terminating...");

Environment.Exit(1);

break;

case 1:

Console.WriteLine("Enter Name of Contact : ");

String tempName = Console.ReadLine()!;

Console.WriteLine("Enter the Phone Number : ");

int tempnumber = int.Parse(Console.ReadLine()!);

AddContact(tempName, tempnumber, names, numbers, ref total );

break;

case 2:

SearchContact(numbers, names, total);

break;

case 3:

Console.WriteLine("Enter Contact Name : ");

String cname = Console.ReadLine()!;

EditContact(names, numbers, cname, ref total);

break;

case 4:

DeleteContact(names, numbers, ref total);

break;

default:

Console.WriteLine("Please Choose between 1 - 4");

break;

} } }

public static void AddContact(String name, int number, String[] names, int[] numbers, ref int total)

{

if (!(total <= 10))

{

Console.WriteLine("Cannot add more contacts!!");

return;

}

names[total] = name;

numbers[total] = number;

Console.WriteLine("Contact " + names[total] + " saved with number : " + numbers[total]);

total++; // increase total count

return;

}

public static void SearchContact(int[] numbers, String[] names , int total) {

if(total == 0) {

Console.WriteLine("Sorry , but there are no contacts as of now!!");

return;

}

Console.WriteLine("On what basis : ");

Console.WriteLine("1)Phone Number \n2)Contact Name");

int ch = int.Parse(Console.ReadLine()!);

switch (ch) {

case 1:

Console.WriteLine("Enter Phone Number : ");

int tempNumber = int.Parse(Console.ReadLine()!);

String holder = SearchByNumber(tempNumber, numbers, names);

Console.WriteLine("The number " + tempNumber + " is held by " + holder);

return;

case 2:

Console.WriteLine("Enter Contact Name : ");

String tempName = Console.ReadLine()!;

int answer = SearchByName(tempName, numbers, names , total);

if(answer == -1) {

Console.WriteLine("Not found!!");

return;

}

Console.WriteLine("Phone number of " + tempName + " is " + answer);

return;

default:

Console.WriteLine("Please Enter a valid choice");

break;

} }

public static String SearchByNumber(int tempNumber, int[] numbers, String[] names) {

for (int i = 0; i < numbers.Length; i++) {

if (numbers[i] == tempNumber) {

return names[i];

} }

return "Not found!!";

}

public static int SearchByName(String tempName, int[] numbers, String[] names, int total) {

for (int i = 0; i < total; i++) {

if (names[i].Equals(tempName, StringComparison.OrdinalIgnoreCase))

{

return numbers[i];

}

}

return -1; // Indicates that the contact was not found

}

public static void EditContact(String[] names, int[] numbers ,String cname ,ref int total) {

int check = SearchByName(cname, numbers, names, total);

if(check == -1) {

Console.WriteLine("Contact dosen't exist!!");

return;

}

Console.WriteLine("Enter new name of the contact : ");

String newName = Console.ReadLine()!;

Console.WriteLine("Enter new number of the contact : ");

int newNumber = int.Parse(Console.ReadLine()!);

for (int i = 0; i < total; i++) {

if (names[i].Equals(cname, StringComparison.OrdinalIgnoreCase))

{

numbers[i] = newNumber; // number changed

names[i] = newName; ; // name changed

Console.WriteLine("Now "+ names[i] + "'s phone number is " + numbers[i]);

return;

} } }

public static void DeleteContact(String[] names, int[] numbers, ref int total) {

Console.WriteLine("Enter the name of the contact to delete: ");

String nameToDelete = Console.ReadLine()!;

int indexToDelete = -1;

for (int i = 0; i < total; i++)

{

if (names[i].Equals(nameToDelete, StringComparison.OrdinalIgnoreCase))

{

indexToDelete = i;

break;

}

}

if (indexToDelete == -1)

{

Console.WriteLine("Contact not found!");

return;

}

for (int i = indexToDelete; i < total - 1; i++) {

names[i] = names[i + 1];

numbers[i] = numbers[i + 1];

}

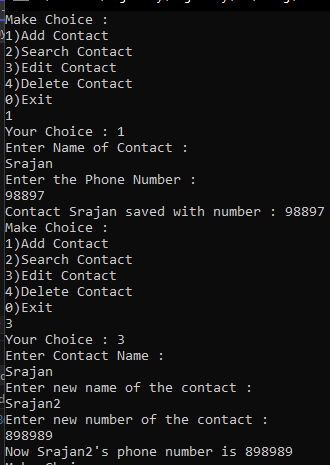
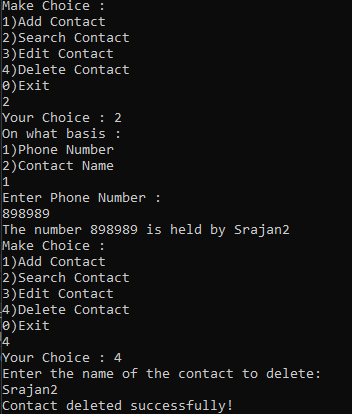
names[total - 1] = null; // setting the last element to null and 0

numbers[total - 1] = 0;

total--;

Console.WriteLine("Contact deleted successfully!");

}

} // end of class

**Q. Bank System Program: with three classes - Bank, Account, SavingsAccount, and CurrentAccount. The bank should have a list of accounts and methods for adding them. Accounts should be an interface with methods to deposit, withdraw, calculate interest, and view balances. SavingsAccount and CurrentAccount should implement the Account interface and have their own unique methods.**

interface IAccount {

void Deposit(double amount);

void Withdraw(double amount);

void ViewBalance();

double CalculateInterest();

}

class Bank {

private SavingsAccount[] savingsAccounts = new SavingsAccount[100];

private CurrentAccount[] currentAccounts = new CurrentAccount[100];

private int savingsCount = 0;

private int currentCount = 0;

private static int accountIdCounter = 1;

public void AddSavingsAccount(string name, double initialDeposit)

{

SavingsAccount newAccount = new SavingsAccount(name, accountIdCounter, initialDeposit);

savingsAccounts[savingsCount++] = newAccount;

Console.WriteLine($"Savings Account created successfully with Account ID: {accountIdCounter}");

accountIdCounter++;

}

public void AddCurrentAccount(string name, double initialDeposit) {

CurrentAccount newAccount = new CurrentAccount(name, accountIdCounter, initialDeposit);

currentAccounts[currentCount++] = newAccount;

Console.WriteLine($"Current Account created successfully with Account ID: {accountIdCounter}");

accountIdCounter++;

}

public void ManageAccount() {

Console.Write("Enter Account ID to manage: ");

int accountId = int.Parse(Console.ReadLine());

bool accountFound = false;

for (int i = 0; i < savingsCount; i++) {

if (savingsAccounts[i].AccountId == accountId) {

accountFound = true;

Console.WriteLine($"\nManaging Savings Account (ID: {accountId}) for {savingsAccounts[i].Name}");

ManageAccountActions(savingsAccounts[i]);

break;

}

}

if (!accountFound) {

for (int i = 0; i < currentCount; i++) {

if (currentAccounts[i].AccountId == accountId) {

accountFound = true;

Console.WriteLine($"\nManaging Current Account (ID: {accountId}) for {currentAccounts[i].Name}");

ManageAccountActions(currentAccounts[i]);

break;

} }

}

if (!accountFound) {

Console.WriteLine("Account not found!");

}

}

private void ManageAccountActions(IAccount account) {

while (true) {

Console.WriteLine("\n1. Deposit");

Console.WriteLine("2. Withdraw");

Console.WriteLine("3. View Balance");

Console.WriteLine("4. Exit Account Management");

Console.Write("Select an option: ");

int option = int.Parse(Console.ReadLine());

switch (option) {

case 1:

Console.Write("Enter amount to deposit: ");

double depositAmount = double.Parse(Console.ReadLine());

account.Deposit(depositAmount);

break;

case 2:

Console.Write("Enter amount to withdraw: ");

double withdrawAmount = double.Parse(Console.ReadLine());

account.Withdraw(withdrawAmount);

break;

case 3:

account.ViewBalance();

break;

case 4:

Console.WriteLine("Exiting Account Management.");

return;

default:

Console.WriteLine("Invalid option. Try again.");

break;

} }

}

public void DisplayAllAccounts() {

Console.WriteLine("\n--- Savings Accounts ---");

for (int i = 0; i < savingsCount; i++)

{

Console.WriteLine($"Account Holder: {savingsAccounts[i].Name}, Balance: {savingsAccounts[i].Balance}, Account ID: {savingsAccounts[i].AccountId}");

}

Console.WriteLine("\n--- Current Accounts ---");

for (int i = 0; i < currentCount; i++)

{

Console.WriteLine($"Account Holder: {currentAccounts[i].Name}, Balance: {currentAccounts[i].Balance}, Account ID: {currentAccounts[i].AccountId}");

}}}

class SavingsAccount : IAccount {

public string Name { get; private set; }

public int AccountId { get; private set; }

public double Balance { get; private set; }

public SavingsAccount(string name, int accountId, double initialDeposit)

{

Name = name;

AccountId = accountId;

Balance = initialDeposit;

}

public void Deposit(double amount) {

Balance += amount;

Console.WriteLine($"Deposited {amount}, new balance is {Balance}");

}

public void Withdraw(double amount) {

if (amount <= Balance) {

Balance -= amount;

Console.WriteLine($"Withdrew {amount}, new balance is {Balance}");

}

else {

Console.WriteLine("Insufficient funds.");

}

}

public void ViewBalance() {

Console.WriteLine($"Balance: {Balance}");

}

public double CalculateInterest() {

return Balance \* 0.04;

}

}

class CurrentAccount : IAccount {

public string Name { get; private set; }

public int AccountId { get; private set; }

public double Balance { get; private set; }

public CurrentAccount(string name, int accountId, double initialDeposit)

{

Name = name;

AccountId = accountId;

Balance = initialDeposit;

}

public void Deposit(double amount) {

Balance += amount;

Console.WriteLine($"Deposited {amount}, new balance is {Balance}");

}

public void Withdraw(double amount)

{

if (amount <= Balance)

{

Balance -= amount;

Console.WriteLine($"Withdrew {amount}, new balance is {Balance}");

}

else {

Console.WriteLine("Insufficient funds.");

}

}

public void ViewBalance() {

Console.WriteLine($"Balance: {Balance}");

}

public double CalculateInterest() {

return Balance \* 0.03;

}

}

class Program {

static void Main(string[] args) {

Bank bank = new Bank();

while (true) {

Console.WriteLine("\n1. Create Savings Account");

Console.WriteLine("2. Create Current Account");

Console.WriteLine("3. Manage Account");

Console.WriteLine("4. Display All Accounts");

Console.WriteLine("5. Exit");

Console.Write("Select an option: ");

int choice = int.Parse(Console.ReadLine());

switch (choice) {

case 1:

Console.Write("Enter Name: ");

string savingsName = Console.ReadLine();

Console.Write("Enter Initial Deposit: ");

double savingsDeposit = double.Parse(Console.ReadLine());

bank.AddSavingsAccount(savingsName, savingsDeposit);

break;

case 2:

Console.Write("Enter Name: ");

string currentName = Console.ReadLine();

Console.Write("Enter Initial Deposit: ");

double currentDeposit = double.Parse(Console.ReadLine());

bank.AddCurrentAccount(currentName, currentDeposit);

break;

case 3:

bank.ManageAccount();

break;

case 4:

bank.DisplayAllAccounts();

break;

case 5:

Console.WriteLine("Exiting the system...");

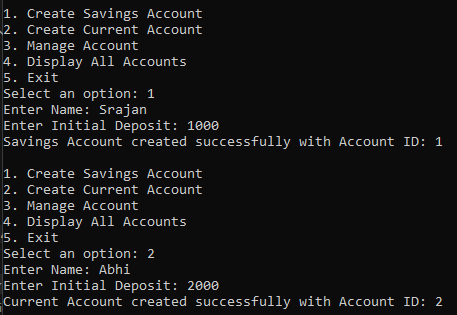
Environment.Exit(1);

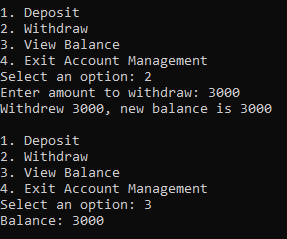
break;

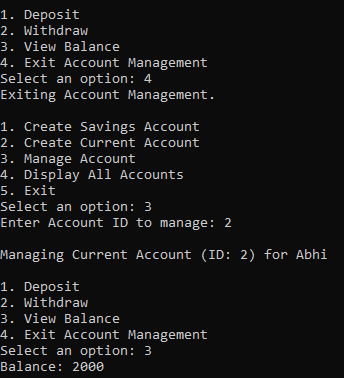
default:

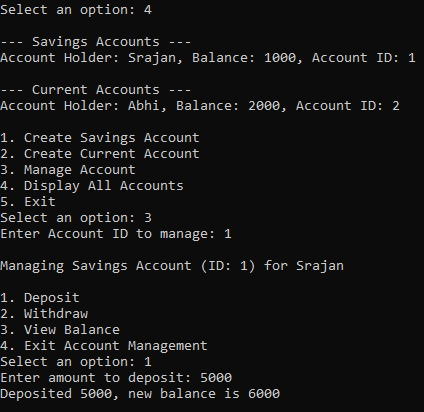
Console.WriteLine("Invalid option. Try again.");

break;

 } } } }



****



**Q. Bus Reservation System**

class BusReservation {

public int[][] Seats;

public static void Main(String[] args) {

BusReservation obj = new BusReservation();

obj.InitializeSeating();

while (true) {

Console.WriteLine("Choose : ");

Console.WriteLine("1)Book Seat ");

Console.WriteLine("2)Cancel Seat Reservation");

Console.WriteLine("3)View Seating Arrangement");

Console.WriteLine("4)Exit");

int choice = int.Parse(Console.ReadLine()!);

switch (choice) {

case 1:

obj.BookSeat();

break;

case 2:

obj.cancelSeat();

break;

case 3:

obj.showSeats();

break;

case 4:

Environment.Exit(1);

break;

default:

Console.WriteLine("Wrong Choice!!");

break;

} }

}

public void InitializeSeating() {

Console.WriteLine("Enter number of rows:");

int numRows = int.Parse(Console.ReadLine()!);

Seats = new int[numRows][];

for (int i = 0; i < numRows; i++) {

Console.WriteLine($"Enter number of seats for row {i + 1}:");

int numSeats = int.Parse(Console.ReadLine()!);

Seats[i] = new int[numSeats];

// Initialize seats as available

for (int j = 0; j < numSeats; j++)

{

Seats[i][j] = 0; // not booked

}

} }

public void BookSeat() {

showSeats();

Console.WriteLine("Enter the row no : ");

int row = int.Parse(Console.ReadLine()!);

Console.WriteLine("Enter the Column no. : ");

int col = int.Parse(Console.ReadLine()!);

if (row >= 0 && row < Seats.Length &&

col >= 0 && col < Seats[row].Length)

{

Seats[row-1][col-1] = 1;

showSeats();

}

else {

Console.WriteLine("Invalid Row or Column!!");

return;

}

}

public void cancelSeat() {

Console.WriteLine("Enter the row no : ");

int row = int.Parse(Console.ReadLine()!);

Console.WriteLine("Enter the Column no. : ");

int col = int.Parse(Console.ReadLine()!);

if (row >= 0 && row < Seats.Length &&

col >= 0 && col < Seats[row].Length) {

if (Seats[row - 1][col - 1] == 1) {

Seats[row - 1][col - 1] = 0;

Console.WriteLine("Seat Successfully Canceled!!");

showSeats(); // show seating arrangement after cancelling

}

else {

Console.WriteLine("Seat is already vacant!!");

return;

}

}

else {

Console.WriteLine("Invalid Row or Column!!");

return;

}

}

public void showSeats() {

for (int i = 0; i < Seats.Length; i++) {

Console.Write($"Row {i+1} : ");

for (int j = 0; j < Seats[i].Length; j++) {

Console.Write(Seats[i][j] + " ");

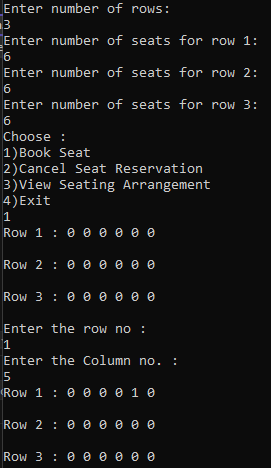
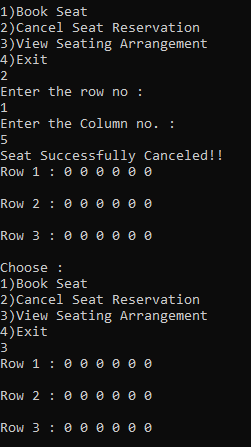
}

Console.WriteLine("\n");

}

}

}



**Q. Enter sales data for four zones and then rank them on the basis of highest sale to lowest.**

class ZoneSales {

public int[][] SalesChart;

public int[] ranks = new int[4];

public static void Main(String[] args) {

ZoneSales obj = new ZoneSales();

obj.Initialize();

obj.printSalesChart();

obj.inputSalesData();

obj.printSalesChart();

obj.rankSales();

}

public void Initialize() {

SalesChart = new int[4][]; // 4 Zones : rows

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 12; j++){ // 12 Months : cols

SalesChart[i] = new int[12];

}

}

}

void printSalesChart() {

for (int i = 0; i < SalesChart.Length; i++) {

Console.Write($"Zone {i} : ");

for (int j = 0; j < 12; j++) { // 12 Months : cols

Console.Write(SalesChart[i][j] + " ");

}

Console.WriteLine("\n");

}

}

void inputSalesData() {

for (int i = 0; i < SalesChart.Length; i++) {

Console.Write($"Zone {i} : ");

for (int j = 0; j < 12; j++) { // 12 Months : cols

SalesChart[i][j] = int.Parse(Console.ReadLine()!);

} Console.WriteLine("\n");

} }

void rankSales() {

for (int i = 0; i < SalesChart.Length; i++) {

int zoneEarnings = 0;

for (int j = 0; j < 12; j++) {

zoneEarnings += SalesChart[i][j];

}

ranks[i] = zoneEarnings;

}

int[] sortedZones = { 0, 1, 2, 3 };

Array.Sort(ranks, sortedZones); // Sort based on earnings

Array.Reverse(sortedZones); // To get in descending order

Console.WriteLine("The Scores (from highest to lowest) are: ");

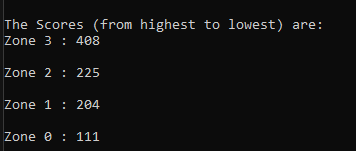
for (int i = 0; i < sortedZones.Length; i++) {

int zoneIndex = sortedZones[i];

Console.Write($"Zone {zoneIndex} : ");

Console.Write(ranks[zoneIndex]); Console.WriteLine("\n");

}}}



**Q. WAP to create a class to store emails, view all email and view emails with .com subdomain**

class Email {

string[] emails;

int ptr;

int length;

Email(int x)

{

length = x;

emails = new string[length];

ptr = 0;

}

bool addEmail(string email)

{

if (ptr == length) return false;

if (!email.Contains('@') || !email.Contains('.')) return false;

emails[ptr++] = email;

return true;

}

void printEmails() {

for (int i = 0; i < ptr; i++)

{

Console.WriteLine(emails[i]);

}

}

void printComEmails() {

for (int i = 0; i < ptr; i++)

{

if(emails[i].EndsWith(".com")) Console.WriteLine(emails[i]);

}

}

public static void Main() {

Email app = new Email(10);

while (true) {

Console.WriteLine("1: Enter Email");

Console.WriteLine("2: View All Emails");

Console.WriteLine("3: View .com Emails");

Console.WriteLine("4: Exit");

Console.Write("Choice: ");

int choice = int.Parse(Console.ReadLine() !);

switch (choice) {

case 1:

Console.WriteLine("Enter your email address: ");

string email = Console.ReadLine() !;

bool res = app.addEmail(email);

if (res == true) Console.WriteLine("Email Added Successfully");

else Console.WriteLine("Email Not Added");

break;

case 2:

app.printEmails();

break;

case 3:

app.printComEmails();

break;

case 4:

Console.WriteLine("ThankYou");

return;

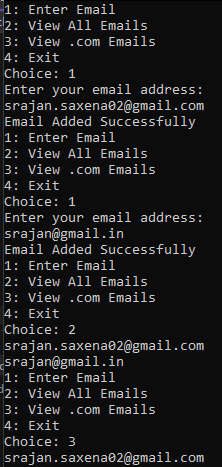
default:

Console.WriteLine("Enter Valid Input");

break;

} } }

}



**Q. Use Inheritance to make a program with classes: Backoffice, Developer, Tester and Market.**

class Employee {

public string Name { get; set; }

public int EmpId { get; set; }

public double BasicSalary { get; set; }

public double Ta { get; set; }

public double Da { get; set; }

public double Hra { get; set; }

public double Ppf { get; set; }

public Employee(string name, int empId, double basicSalary) {

Name = name;

EmpId = empId;

BasicSalary = basicSalary;

Ta = basicSalary \* 0.10; // assuming percentage

Da = basicSalary \* 0.05; // 5 % of salary

Hra = basicSalary \* 0.15; // 15 % of salary

Ppf = basicSalary \* 0.08; // 8 % of salary

}

public virtual double CalculateSalary() {

return BasicSalary + Ta + Da + Hra - Ppf;

}

public virtual void DisplaySalarySlip() {

Console.WriteLine($"Name: {Name}, Emp ID: {EmpId}, Total Salary: {CalculateSalary()}");

}

}

class Developer : Employee {

public double Incentive { get; set; }

public Developer(string name, int empId, double basicSalary, double incentive): base(name, empId, basicSalary)

{

Incentive = incentive;

}

public override double CalculateSalary() {

return base.CalculateSalary() + Incentive;

}

public override void DisplaySalarySlip() {

base.DisplaySalarySlip();

Console.WriteLine($"Incentive: {Incentive}, Net Salary: {CalculateSalary()}");

}

}

class Backoffice : Employee {

public double Incentive { get; set; }

public Backoffice(string name, int empId, double basicSalary, double incentive): base(name, empId, basicSalary)

{

Incentive = incentive;

}

public override double CalculateSalary() {

return base.CalculateSalary() + Incentive;

}

public override void DisplaySalarySlip() {

base.DisplaySalarySlip();

Console.WriteLine($"Incentive: {Incentive}, Net Salary: {CalculateSalary()}");

}

}

class Tester : Employee {

public double ProjectIncentive { get; set; }

public double ReportIncentive { get; set; }

public int NumberOfProjects { get; set; }

public int NumberOfReports { get; set; }

public Tester(string name, int empId, double basicSalary, int numberOfProjects, int numberOfReports): base(name, empId, basicSalary)

{

NumberOfProjects = numberOfProjects;

NumberOfReports = numberOfReports;

ProjectIncentive = 100 \* numberOfProjects; // Example calculation

ReportIncentive = 50 \* numberOfReports; // Example calculation

}

public override double CalculateSalary() {

return base.CalculateSalary() + ProjectIncentive + ReportIncentive;

}

public override void DisplaySalarySlip()

{

base.DisplaySalarySlip();

Console.WriteLine($"Project Incentive: {ProjectIncentive}, Report Incentive: {ReportIncentive}, Net Salary: {CalculateSalary()}");

}

}

class Market : Employee{

public double Incentive { get; set; }

public Market(string name, int empId, double basicSalary, double incentive): base(name, empId, basicSalary)

{

Incentive = incentive;

}

public override double CalculateSalary() {

return base.CalculateSalary() + Incentive;

}

public override void DisplaySalarySlip() {

base.DisplaySalarySlip();

Console.WriteLine($"Incentive: {Incentive}, Net Salary: {CalculateSalary()}");

}

}

class Program {

static void Main(string[] args) {

Console.WriteLine("Menu:");

Console.WriteLine("1. Developer");

Console.WriteLine("2. Backoffice");

Console.WriteLine("3. Tester");

Console.WriteLine("4. Market");

Console.WriteLine("Enter your choice (1-4): ");

int choice = int.Parse(Console.ReadLine()!);

switch (choice) {

case 1:

Console.WriteLine("Enter Developer Details:");

Developer developer = new Developer("Alice", 101, 50000, 3000);

developer.DisplaySalarySlip();

break;

case 2:

Console.WriteLine("Enter Backoffice Details:");

Backoffice backoffice = new Backoffice("Bob", 102, 40000, 2000);

backoffice.DisplaySalarySlip();

break;

case 3:

Console.WriteLine("Enter Tester Details:");

Tester tester = new Tester("Charlie", 103, 45000, 5, 10);

tester.DisplaySalarySlip();

break;

case 4:

Console.WriteLine("Enter Market Details:");

Market market = new Market("David", 104, 55000, 4000);

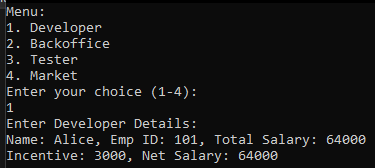
market.DisplaySalarySlip();

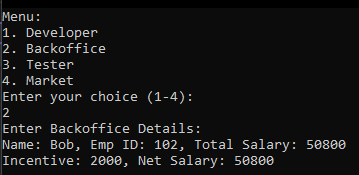
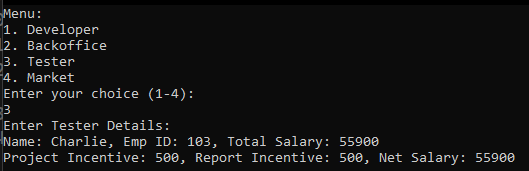
break;

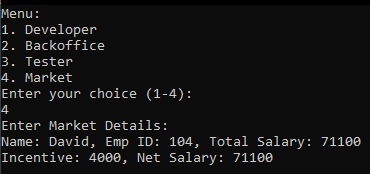
default:

Console.WriteLine("Invalid choice!");

break;

 } } }





**Q. WAP to enter a string, convert it to uppercase and add 2 in each character present in the string and display original and modified string.**

class Program {

public static void Main(string[] args) {

Console.WriteLine("Enter a string : ");

string x = Console.ReadLine()!;

Console.WriteLine($"The original String : {x}");

x = x.ToUpper();

string ans = "";

for (int i = 0; i < x.Length; i++) {

if (x[i] == ' ') {

ans += ' '; // keep spaces consistent

continue;

}

if (x[i] + 2 > 'Z') {

ans += (char)((x[i] + 2) - 26);

}

else {

ans += (char)(x[i] + 2);

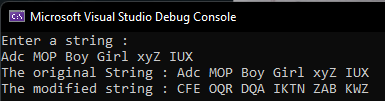
}

}

Console.WriteLine($"The modified string : {ans}");

}

}



**Q. Input a name, convert each character to its opposite case and print the original and modified names along with suitable messages.**

class Program {

public static void Main(string[] args) {

Console.WriteLine("Enter name : ");

string x = Console.ReadLine()!;

string result = "";

for (int i = 0; i < x.Length; i++) {

if (char.IsUpper(x[i])) {

result += char.ToLower(x[i]);

}

else if (char.IsLower(x[i])) {

result += char.ToUpper(x[i]);

}

else {

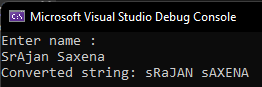
result += x[i]; // keeping non-alpha chars unchanged

} }

Console.WriteLine("Converted string: " + result);

}

}



**Q. WAP to accept a word and convert to lowercase and display new word by replacing only the vowels with the character that follows it.**

class Program {

public static void Main(string[] args) {

Console.WriteLine("Enter a string : ");

string x = Console.ReadLine()!;

x = x.ToLower();

string ans = "";

for (int i = 0; i < x.Length; i++) {

if (x[i] == 'a' || x[i] == 'e' || x[i] == 'i' || x[i] == 'o' || x[i] == 'u')

{

ans += (char)(x[i] + 1);

}

else {

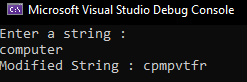
ans += x[i];

} }

Console.WriteLine($"Modified String : {ans}");

}

}



**Q. WAP to create a class ArrangeLetters and method Arrage() with a string parameter. Convert the word in uppercase and arrange each letter of the word in alphabetical order. Print the word before and after arranging the letters in A-Z order.**

class ArrangeLetters {

public static void Main(string[] args) {

Arrange("bca");

}

public static void Arrange(string x) {

Console.WriteLine($"Given : {x}");

x = x.ToUpper();

Console.WriteLine($"In Upper Case : {x}");

char[] arr = x.ToCharArray();

for(int i = 0; i < arr.Length - 1; i++) {

for(int j = 0; j < arr.Length - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

char temp = arr[j]; // swap

arr[j] = arr[j + 1];

arr[j + 1] = temp;

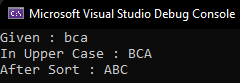
} } }

String ans = new String(arr);

Console.WriteLine($"After Sort : {ans}");

}

}



**Q. WAP to create an interface Encryptable with methods encrypt (String data) and decrypt (String encryptedData) that define encryption and decryption operations. Create a class AES that implements the Encryptable interface and provide their own encryption and decryption algorithms.**

public interface Encryptable {

string Encrypt(string data);

string Decrypt(string encryptedData);

}

public class AES : Encryptable {

private int shift;

public AES(int shiftValue) {

shift = shiftValue;

}

public string Encrypt(string data) {

char[] encrypted = new char[data.Length];

for (int i = 0; i < data.Length; i++) {

char c = data[i];

if (char.IsLetter(c)) {

char d = char.IsUpper(c) ? 'A' : 'a';

encrypted[i] = (char)((c + shift - d) % 26 + d);

}

else {

encrypted[i] = c;

} }

return new string(encrypted);

}

public string Decrypt(string encryptedData) {

char[] decrypted = new char[encryptedData.Length];

for (int i = 0; i < encryptedData.Length; i++) {

char c = encryptedData[i];

if (char.IsLetter(c)){

char d = char.IsUpper(c) ? 'A' : 'a';

decrypted[i] = (char)((c - shift - d + 26) % 26 + d);

}

else {

decrypted[i] = c;

} }

return new string(decrypted);

}}

class Program {

static void Main(string[] args) {

while (true) {

Console.WriteLine("\n--- Encryption Program ---");

Console.WriteLine("1. Encrypt using AES (Shift Cipher)");

Console.WriteLine("2. Decrypt using AES (Shift Cipher)");

Console.WriteLine("3. Exit");

Console.Write("Choose an option: ");

int choice = int.Parse(Console.ReadLine()!);

if (choice == 1) {

Console.Write("Enter text to encrypt: ");

string text = Console.ReadLine()!;

Console.Write("Enter shift value: ");

int shift = int.Parse(Console.ReadLine()!);

AES aes = new AES(shift);

string encryptedText = aes.Encrypt(text);

Console.WriteLine("Encrypted Text: " + encryptedText);

}

else if (choice == 2)

{

Console.Write("Enter text to decrypt: ");

string text = Console.ReadLine()!;

Console.Write("Enter shift value: ");

int shift = int.Parse(Console.ReadLine()!);

AES aes = new AES(shift);

string decryptedText = aes.Decrypt(text);

Console.WriteLine("Decrypted Text: " + decryptedText);

}

else if (choice == 3) {

Console.WriteLine("Exiting program...");

break;

}

else {

Console.WriteLine("Invalid choice, try again.");

} } } }