

Contents

Assignment 1

- 1.1** Problem
- 1.2** Toward Solution [Abstract Approach]
- 1.3** Problem Solution [Code/Program]
- 1.4** Program OutPut
- 1.5** Program/Code Explanation

Page **1-4**

Assignment 3

- 3.1** Problem
- 3.2** Toward Solution [Abstract Approach]
- 3.3** Problem Solution [Code/Program]
- 3.4** Program OutPut
- 3.5** Program/Code Explanation

Page **8-11**

Assignment 2

- 2.1** Problem
- 2.2** Toward Solution [Abstract Approach]
- 2.3** Problem Solution [Code/Program]
- 2.4** Program OutPut
- 2.5** Program/Code Explanation

Page **4-8**

Assignment 4

- 4.1** Problem
- 4.2** Toward Solution [Abstract Approach]
- 4.3** Problem Solution [Code/Program]
- 4.4** Program OutPut
- 4.5** Program/Code Explanation

Page **12-15**

Assignment 5

- 5.1** Problem
- 5.2** Toward Solution [Abstract Approach]
- 5.3** Problem Solution [Code/Program]
- 5.4** Program OutPut
- 5.5** Program/Code Explanation

Page **16-20**

Assignment 1

```

void Display() {
    cout << "The User name is " << Name << endl;
    cout << "Total Unit Consumed " << UnitConsumed << endl;
    cout << "Total Bill After All Deaduction " << TotalAmount << endl;
}
};
int main() {
    int UserSize;
    string name;
    float unit;

    cout << "Enter the total Number of user: ";
    cin >> UserSize;
    cin.clear();
    cin.ignore();

    CalculEBill* User = new CalculEBill[UserSize];

    for (int i = 0; i < UserSize; ++i) {
        cout << "Enter the User Name: ";
        getline(cin, name);

        cout << "Enter the user " << name << " unit: ";
        cin >> unit;

        cin.clear();
        cin.ignore();

        User[i] = CalculEBill(name, unit);
    }

    for (int i = 0; i < UserSize; ++i) {
        User[i].Display();
        cout << "<----->" << endl;
    }

    delete[] User;

    return 0;
}

```

1.4 Program OutPut :

```

C:\Practical_Paper\CPPs_C++> ls
./ ./ As_1/ As_2/ As_3/ As_4/ As_5/ *MCA C++ Lab Assignments.pdf* .vscode/

C:\Practical_Paper\CPPs_C++> cd As_1/

C:\Practical_Paper\CPPs_C++\As_1> ls
./ ./ ElectBill\ ElectBill.cpp .vscode/

C:\Practical_Paper\CPPs_C++\As_1> ./ElectBill
Enter the total Number of user: 3
Enter the User Name: Md Afsarul Hoque
Enter the user Md Afsarul Hoque unit: 200
Enter the User Name: Sk Sanir Rahaman
Enter the user Sk Sanir Rahaman unit: 20
Enter the User Name: Aditi
Enter the user Aditi unit: 100
The User name is Md Afsarul Hoque
Total Unit Consumed 200
Total Bill After All Deaduction 140
<----->
The User name is Sk Sanir Rahaman
Total Unit Consumed 20
Total Bill After All Deaduction 50
<----->
The User name is Aditi
Total Unit Consumed 100
Total Bill After All Deaduction 60
<----->

C:\Practical_Paper\CPPs_C++\As_1>

C:\Practical_Paper\CPPs_C++\As_1> ./ElectBill
Enter the total Number of user: 2
Enter the User Name: Md Alex
Enter the user Md Alex unit: 332
Enter the User Name: Xyz
Enter the user Xyz unit: 10
The User name is Md Alex
Total Unit Consumed 332
Total Bill After All Deaduction 248.8
<----->
The User name is Xyz
Total Unit Consumed 10
Total Bill After All Deaduction 50
<----->

C:\Practical_Paper\CPPs_C++\As_1>

```

The Above Output We See That Our Program is work Well

Assignment 1

1.5 Program/Code Explanation :

```
#include <iostream>
#include <string>
using namespace std;
```

- This part includes necessary header files for input/output operations (iostream) and string manipulation (string).
- using namespace std; declares that we are going to use symbols from the std namespace, which includes commonly used functionality like cin, cout, string, etc.

```
class CalculeBill {
    string Name;
    float UnitConsumed, TotalAmount;
public:
```

- This declares a class named CalculeBill. It has three private data members: Name, UnitConsumed, and TotalAmount.
- public: declares that the members following it (CalculeBill() constructor, CalculeBill(string name, float unitConsumed) constructor, FinalBill(float& totalAmount), calculateBill(float unit), and Display()) are accessible outside the class.

```
CalculeBill():
    Name(""),
    UnitConsumed(0.0),
    TotalAmount(0.0){
}
```

- This is a constructor of the class CalculeBill with no arguments. It initializes the Name to an empty string, UnitConsumed to 0.0, and TotalAmount to 0.0.

```
CalculeBill(string name, float unitConsumed):
    Name(name), UnitConsumed(unitConsumed),
    TotalAmount(0.0){
    calculateBill(UnitConsumed);
}
```

- This is a constructor of the class CalculeBill with two arguments (name and unitConsumed). It initializes Name with the provided name, UnitConsumed with the provided unitConsumed, and TotalAmount to 0.0.
- It then calls the calculateBill function to calculate the total bill based on the consumed units.

```
void FinalBill(float& totalAmount){
    if (totalAmount < 50){
        totalAmount = 50;
    }
    if (totalAmount > 300){
        totalAmount += (totalAmount * 15 / 100);
    }
}
```

- This function FinalBill calculates the final bill amount after applying some rules. If the total amount is less than 50, it sets it to 50. If it's greater than 300, it adds 15% to the total amount.

```
void calculateBill(float unit){
    if (unit <= 100){
        TotalAmount = (unit * 0.6);
        FinalBill(TotalAmount);
    } else if (unit > 100 && unit <= 300){
        TotalAmount = ((100 * 0.6) + ((unit - 100) * 0.8));
        FinalBill(TotalAmount);
    } else if (unit > 300){
        TotalAmount = (((100 * 0.6) + (200 * 0.8)) + ((unit - 300) * 0.9));
        FinalBill(TotalAmount);
    }
}
```

- This function calculateBill calculates the total bill based on the consumed units (unit). It applies different rates based on different usage levels.

Assignment 1

```
void Display() {
    cout << "The User name is " << Name << endl;
    cout << "Total Unit Consumed " << UnitConsumed << endl;
    cout << "Total Bill After All Deduction " << TotalAmount << endl;
}
```

- This function Display displays the user's name, total units consumed, and the final bill amount after all deductions.

```
int main() {
    int UserSize;
    string name;
    float unit;

    cout << "Enter the total Number of user: ";
    cin >> UserSize;
    cin.clear();
    cin.ignore();

    CalculEBill* User = new CalculEBill[UserSize];

    for (int i = 0; i < UserSize; ++i) {
        cout << "Enter the User Name: ";
        getline(cin, name);

        cout << "Enter the user " << name << " unit: ";
        cin >> unit;

        cin.clear();
        cin.ignore();

        User[i] = CalculEBill(name, unit);
    }
```

- This is the main() function. It prompts the user to enter the total number of users and stores it in UserSize.
- It then dynamically allocates memory for an array of CalculEBill objects using the new operator.
- It then iterates through each user, prompting for their name and units consumed, and creates CalculEBill objects for each user with the entered data

```
for (int i = 0; i < UserSize; ++i) {
    User[i].Display();
    cout << "<----->" << endl;
}
```

- After creating all CalculEBill objects, it displays the details of each user by calling the Display function for each object.

```
delete[] User;
return 0;
}
```

- Finally, it deallocates the memory allocated for the array of CalculEBill objects using the delete[] operator and returns '0' to indicate successful execution of the program.h object.

Assignment 2

2.1 Problem :

Write a C program to take two strings as input and perform the following operations:

- Find the length of the strings using pointers.
- Compare the two strings using pointers to determine whether both are same or not.
- Concatenate the two strings using pointers.

2.2 Toward Solution [Abstract Approach]

Problem Definition:

- Perform various string operations: Calculate string lengths
- Compare strings (case-insensitive)
- Concatenate strings

[illegible]

- Break down the problem into smaller functions: Function to calculate string length
- Function to compare two strings
- Function to concatenate two strings
- Define a function to handle user input and output.

Main Program:

- Call the input/output function to prompt for strings and display results.

Specific Functions:

- Implement a function to calculate string length: Iterate through the string until the null terminator is reached.
- Implement a function to compare strings: Check for length differences first.
- If lengths are equal, compare characters case-insensitively.
- Implement a function to concatenate strings: Allocate memory for the combined string.
- Copy the first string into the result.
- Append the second string to the result.
- Deallocate memory when finished.

2.3 Problem Solution [Code/Program]:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

void strCompare(const char *str, const char *str2);
void stringConcat(const char *str, const char *str2);
void getstring();
int stringLength(const char *str);

void getstring() {
    char string[250], string2[250];
    printf("Please Enter the String 1: ");
    scanf("%s", string);
    printf("Please Enter the String 2: ");
    scanf("%s", string2);
    printf("The String 1 is: \" %s \" and and Lenght is : %d \n", string , stringLength(string));
    printf("The String 2 is: \" %s \" and and Lenght is : %d \n", string2 , stringLength(string2));
    printf("_____Compare Strings _____\n");
    strCompare(string, string2);
    printf("_____String Concatenation _____\n");
    stringConcat(string, string2);
}

int stringLength(const char *str) {
    const char *ptr = str;
    while (*ptr != '\0') {
        ptr++;
    }
    return (ptr - str);
}

void strCompare(const char *str, const char *str2) {
    int strlen1, strlen2;
    strlen1 = stringLength(str);
    strlen2 = stringLength(str2);
    if (strlen1 > strlen2) {
        printf(" \"%s\" is longer than \" %s\" and these two are not Equal\n", str, str2);
    }
    else if (strlen1 < strlen2) {
        printf(" \"%s\" is shorter than \"%s\" and these two are not Equal\n", str, str2);
    }
    else {
        while (*str != '\0' && *str2 != '\0') {
            if (tolower(*str) == tolower(*str2)) {
                str++;
                str2++;
            }
            else {
                printf("Strings are of equal length but not equal in characters.\n");
                return;
            }
        }
        printf("Strings are of equal length and equal in characters ie Both are Same.\n");
    }
}
```

Assignment 2

```

void stringConcat(const char *str, const char *str2){
    int strlen1, strlen2;
    strlen1 = stringLength(str);
    strlen2 = stringLength(str2);

    char *result = (char *)malloc(strlen1 + strlen2 + 1);
    if (result == NULL){
        printf("Memory allocation failed.\n");
        return;
    }

    strcpy(result, str);
    strcat(result, str2);
    printf("The concatenation of string \"%s\" and string \"%s\" is: \"%s\"\n", str, str2, result);

    free(result); // Free the dynamically allocated memory
}

int main(){
    getstring();
    return 0;
}

```

2.4 Program OutPut :

```

C:\Practical_Paper\OOPs_C++\AS_2> gcc srtingoper.c
C:\Practical_Paper\OOPs_C++\AS_2> ./srtingoper
Please Enter the String 1: HIiii
Please Enter the String 2: Melooo
The String 1 is: "HIiii" and and Lenght is : 5
The String 2 is:"Melooo" and and Lenght is : 6
-----Compare Strings-----
"HIiii" is shorter than "Melooo" and these two are not Equal
-----String Concatenation-----
The concatenation of string "HIiii" and string "Melooo" is: "HIiiiMelooo"

C:\Practical_Paper\OOPs_C++\AS_2> ./srtingoper
Please Enter the String 1: HIIIII
Please Enter the String 2: HIIIII
The String 1 is: "HIIIII" and and Lenght is : 5
The String 2 is:"HIIIII" and and Lenght is : 5
-----Compare Strings-----
Strings are of equal length and equal in characters ie Both are Same.
-----String Concatenation-----
The concatenation of string "HIIIII" and string "HIIIII" is: "HIIIIHHIIII"

C:\Practical_Paper\OOPs_C++\AS_2> ./srtingoper
Please Enter the String 1: HIiii
Please Enter the String 2: hIii
The String 1 is: "HIiii" and and Lenght is : 4
The String 2 is:"hIii" and and Lenght is : 4
-----Compare Strings-----
Strings are of equal length and equal in characters ie Both are Same.
-----String Concatenation-----
The concatenation of string "HIiii" and string "hIii" is: "HIiihIii"

```

The Above Output We See That Our Program is work Well

2.5 Program/Code Explanation :

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

```

These lines include necessary header files for input/output operations (**stdio.h**), string manipulation (**string.h**), memory allocation (**stdlib.h**), and character type functions (**ctype.h**).

```

void strCompare(const char *str, const char *str2);
void stringConcat(const char *str, const char *str2);
void getstring();
int stringLength(const char *str);

```

- These lines are function prototypes for **strCompare**, **stringConcat**, **getstring**, and **stringLength**. They declare functions that will be defined later in the program

```

void getstring(){
    char string[250], string2[250];
    printf("Please Enter the String 1: ");
    scanf("%s", string);
    printf("Please Enter the String 2: ");
    scanf("%s", string2);
}

```

Assignment 2

```
printf("The String 1 is: \" %s \" and Length is : %d \n", string , strlen(string));
printf("The String 2 is: \" %s \" and Length is : %d \n", string2 , strlen(string2));
printf("_____Compare Strings _____\n");
strcmp(string, string2);
printf("_____String Concatenation _____\n");
strcat(string, string2);
}
```

- This function `getstring` prompts the user to enter two strings, reads them using `scanf`, and then prints the length of each string.
- It then calls `strcmp` to compare the two strings and `strcat` to concatenate them.

```
int strlen(const char *str) {
    const char *ptr = str;
    while (*ptr != '\0') {
        ptr++;
    }
    return (ptr - str);
}
```

- This function **`strlen`** calculates the length of a string by iterating through it until it encounters the null terminator `\0`.

```
void strcmp(const char *str, const char *str2) {
    int strlen1, strlen2;
    strlen1 = strlen(str);
    strlen2 = strlen(str2);

    if (strlen1 > strlen2) {
        printf(" \"%s\" is longer than \" %s\" and these two are not Equal\n", str, str2);
    }
    else if (strlen1 < strlen2) {
        printf(" \"%s\" is shorter than \" %s\" and these two are not Equal\n", str, str2);
    }
    else {
        while (*str != '\0' && *str2 != '\0') {
            if (tolower(*str) == tolower(*str2)) {
                str++;
                str2++;
            }
            else {
                printf("Strings are of equal length but not equal in characters.\n");
                return;
            }
        }
        printf("Strings are of equal length and equal in characters ie Both are Same.\n");
    }
}
```

- This function `strcmp` compares two strings character by character. It first compares the lengths of the strings. If the lengths are different, it prints a message accordingly.
- If the lengths are the same, it compares each character of the strings while ignoring case differences using `tolower`.
- If any character mismatch is found, it prints a message indicating that the strings are not equal.
- If all characters match, it prints a message indicating that the strings are equal.

```
void strconcat(const char *str, const char *str2) {
    int strlen1, strlen2;
    strlen1 = strlen(str);
    strlen2 = strlen(str2);

    char *result = (char *)malloc(strlen1 + strlen2 + 1);
    if (result == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }
}
```


Assignment 2

```
strcpy(result, str);
strcat(result, str2);
printf("The concatenation of string \"%s\" and string \"%s\" is: \"%s\"\\n", str, str2, result);

free(result); // Free the dynamically allocated memory
}
```

- This function stringConcat concatenates two strings and prints the result. It first calculates the lengths of the two strings.
- It then allocates memory for a new string to hold the concatenated result using malloc.
- If memory allocation fails, it prints an error message and returns.
- If allocation succeeds, it copies the first string into the result using strcpy, then appends the second string using strcat.
- After printing the concatenated result, it frees the dynamically allocated memory using free.

```
int main() {
    getstring();
    return 0;
}
```

- This is the main function, which is the entry point of the program. It simply calls the getstring function and returns 0 to indicate successful execution.

Assignment 3

3.1 Problem :

Create a Structure called employee with the following details as variables within it.

- Employee Id
- Name of the employee
- Age
- Designation
- Salary

Write a C program to create array of 5 objects for the structure to access these and print the employee id, name, age, designation and salary.

3.2 Toward Solution [Abstract Approach]

Problem Definition:

- Manage employee data, including ID, name, age, designation, and salary.
- Allow users to create, view, and search for employee information.

Data Structure:

- Define a custom structure (e.g., Employee) to encapsulate employee data.

Main Program:

- Prompt the user for the number of employees to create.
- Dynamically allocate an array of employee structures.
- Get employee details from the user.
- Display a menu of options for the user to interact with data.
- Handle user choices (view all employees, search by ID, exit).
- Deallocate memory when finished.

Specific Functions:

- Create a function to get employee details from the user.
- Create a function to display data for all employees.
- Create a function to search for an employee by ID and display their details

User Interaction:

- Present a clear menu of options to the user.
- Handle invalid user choices gracefully.

3.3 Problem Solution [Code/Program] :

```
#include <iostream>
#include <string>
using namespace std;
struct Employee {
    int employeeId;
    string name;
    int age;
    string designation;
    float salary;
    Employee() : employeeId(0), name(""), age(0), designation(""), salary(0.0f) {}
};
```

Assignment 3

```

int main() {
    int numEmployees;

    cout << "Enter the number of employees you want to create: ";
    cin >> numEmployees;

    // Dynamically allocate an array of Employee objects to avoid stack overflow
    Employee *employees = new Employee[numEmployees];

    for (int i = 0; i < numEmployees; ++i) {
        cout << "\nEnter details for employee " << i + 1 << ": " << endl;

        cout << "Employee ID: ";
        cin >> employees[i].employeeId;

        cout << "Name: ";
        cin.ignore(); // Clear input buffer if needed
        getline(cin, employees[i].name);

        cout << "Age: ";
        cin >> employees[i].age;

        cout << "Designation: ";
        cin.ignore(); // Clear input buffer if needed
        getline(cin, employees[i].designation);

        cout << "Salary: ";
        cin >> employees[i].salary;
    }

    // Display a clear menu for user choice
    int choice;
    cout << "\nChoose an action: \n"
        << "1. Show data for all employees\n"
        << "2. Show data for a specific employee by ID\n"
        << "3. Exit\n";
    cin >> choice;

    while (choice != 3) {
        switch (choice) {
            case 1: // Show data for all employees
                cout << "\nEmployee Details:" << endl;
                for (int i = 0; i < numEmployees; ++i) {
                    cout << "Employee " << i + 1 << ": " << endl;
                    cout << "Employee Id: " << employees[i].employeeId << endl;
                    cout << "Name: " << employees[i].name << endl;
                    cout << "Age: " << employees[i].age << endl;
                    cout << "Designation: " << employees[i].designation << endl;
                    cout << "Salary: " << employees[i].salary << endl << endl;
                }
                break;
            case 2: // Show data for a specific employee by ID
                int searchId;
                cout << "\nEnter the employee ID you want to search for: ";
                cin >> searchId;
                for (int i = 0; i < numEmployees; ++i) {
                    if (employees[i].employeeId == searchId) {
                        cout << "\nEmployee Details:" << endl;
                        cout << "Employee ID: " << employees[i].employeeId << endl;
                        cout << "Name: " << employees[i].name << endl;
                        cout << "Age: " << employees[i].age << endl;
                        cout << "Designation: " << employees[i].designation << endl;
                        cout << "Salary: " << employees[i].salary << endl;
                        break; // Exit the loop after finding the employee
                    }
                }
                else {
                    cout << "You Enter Wrong Employee Id ";
                }
            }

            break;
        default:
            cout << "Invalid choice. Please try again." << endl;
        }
    }
}

```

Assignment 3

```

cout << "\nChoose an action:\n"
<< "1. Show data for all employees\n"
<< "2. Show data for a specific employee by ID\n"
<< "3. Exit\n";
cin >> choice;
}

delete[] employees;

return 0;
}

```

3.4 Program Output :

```

./list_Sem/Practical_Paper/00Ps_C++ >> g++116 ls
./ ./ ./ As_1/ As_2/ As_3/ As_4/ As_5/ 'MCA C++ Lab Assignments.pdf' ./vscode/

./list_Sem/Practical_Paper/00Ps_C++ >> g++116 cd As_3/

./Practical_Paper/00Ps_C++/AS_3 >> g++116 ls
./ ./ ./ Struc* Struc.cpp

./Practical_Paper/00Ps_C++/AS_3 >> g++116 ./Struc
Enter the number of employees you want to create: 2

Enter details for employee 1:
Employee ID: 1
Name: Md Alex
Age: 22
Designation: CEO
Salary: 20000

Enter details for employee 2:
Employee ID: 2
Name: Xyz
Age: 21
Designation: CHD
Salary: 17000

Choose an action:
1. Show data for all employees
2. Show data for a specific employee by ID
3. Exit
1

Employee Details:
Employee 1:
Employee Id: 1
Name: Md Alex
Age: 22
Designation: CEO
Salary: 20000

Employee 2:
Employee Id: 2
Name: Xyz
Age: 21
Designation: CHD
Salary: 17000

Choose an action:
1. Show data for all employees
2. Show data for a specific employee by ID
3. Exit
2

Enter the employee ID you want to search for: 2
You Enter Wrong Employee Id
Employee Details:
Employee ID: 2
Name: Xyz
Age: 21
Designation: CHD
Salary: 17000

Choose an action:
1. Show data for all employees
2. Show data for a specific employee by ID
3. Exit
3

./Practical_Paper/00Ps_C++/AS_3 >> g++116

```

The Above Output We See That Our Program is work Well

3.5 Program/Code Explanation :

```

#include <iostream>
#include <string>

```

These lines include necessary header files for input/output operations (iostream) and string manipulation (string).

```

using namespace std;

```

This line declares that symbols from the std namespace will be used, which includes commonly used functionality like cin, cout, string, etc.

```

struct Employee {
    int employeeId;
    string name;
    int age;
    string designation;
    float salary;
    Employee(): employeeId(0), name(""), age(0), designation(""), salary(0.0f) {}
};

```

Assignment 3

- This defines a structure named Employee which represents the attributes of an employee such as employeeid, name, age, designation, and salary.
- A default constructor is provided to initialize the members of Employee struct with default values.

```
int main() {
    int numEmployees;
    cout << "Enter the number of employees you want to create: ";
    cin >> numEmployees;

    Employee *employees = new Employee[numEmployees];
```

- This is the main() function, the entry point of the program.
- It prompts the user to enter the number of employees they want to create and stores the value in numEmployees.
- Then, it dynamically allocates an array of Employee objects of size numEmployees using the new operator.

```
for (int i = 0; i < numEmployees; ++i) {
    cout << "\nEnter details for employee " << i + 1 << " : " << endl;
    cout << "Employee ID: ";
    cin >> employees[i].employeeid;
    cout << "Name: ";
    cin.ignore(); // Clear input buffer if needed
    getline(cin, employees[i].name);
    cout << "Age: ";
    cin >> employees[i].age;
    cout << "Designation: ";
    cin.ignore(); // Clear input buffer if needed
    getline(cin, employees[i].designation);
    cout << "Salary: ";
    cin >> employees[i].salary;
}
```

- This loop iterates numEmployees times, prompting the user to enter details for each employee.
- It takes input for each member of the Employee struct such as employeeid, name, age, designation, and salary.

```
int choice;
cout << "\nChoose an action:\n"
    << "1. Show data for all employees\n"
    << "2. Show data for a specific employee by ID\n"
    << "3. Exit\n";
cin >> choice;

while (choice != 3) {
    switch (choice) {
        case 1: // Show data for all employees
            // Display details of all employees
            break;
        case 2: // Show data for a specific employee by ID
            // Display details of a specific employee by searching with ID
            break;
        default:
            cout << "Invalid choice. Please try again." << endl;
    }
}
```

- This part displays a menu to the user with three options: to show data for all employees, show data for a specific employee by ID, or exit.
- It uses a while loop to repeatedly prompt the user for their choice until they choose to exit.
- Inside the loop, a switch statement is used to perform different actions based on the user's choice.
- Currently, the cases for showing data for all employees and for a specific employee are not implemented yet.

```
delete[] employees;
return 0;
}
```

Finally, the dynamically allocated memory for the array of Employee objects is released using the delete[] operator, and the program returns 0 to indicate successful execution.

4.3 Problem Solution [Code/Program] :

Assignment 4

```

calculatePercentage();
}

void show() const {
cout << "Student ID: " << studId << endl;
cout << "Name: " << name << endl;
cout << "Marks for 5 subjects:\n";
for (int i = 0; i < 5; ++i) {
cout << sub[i] << ": " << marks[i] << endl;
}

cout << "Percentage: " << percentage << "%" << endl;
}

float getPercentage() const {
return percentage;
}
};

int main() {
int N;
cout << "Enter the number of students: ";
cin >> N;
Student students[N];
for (int i = 0; i < N; ++i) {
cout << "\nEnter details for Student " << i + 1 << ":\n";
students[i].getData();
}
float highestPercentage = 0.0;
int indexHighestPercentage = 0;
for (int i = 0; i < N; ++i) {
if (students[i].getPercentage() > highestPercentage) {
highestPercentage = students[i].getPercentage();
indexHighestPercentage = i;
}
}

cout << "\nDetails of the student with the highest percentage:\n";
students[indexHighestPercentage].show();

return 0;
}

```

4.4 Program OutPut :

```

-- Konsole
New Tab Split View
~/Practical_Paper/00Pr_C++ B1:48 ls
./ ./ As_1/ AS_2/ AS_3/ As_4/ As_5/ 'MCA C++ Lab Assignments.pdf' .vscode/
~/Practical_Paper/00Pr_C++ B1:48 cd As_4/
~/Practical_Paper/00Pr_C++/As_4 B1:48 ls
./ ./ studentresulr studentresulr.cpp
~/Practical_Paper/00Pr_C++/As_4 B1:48 ./studentresulr
Enter the number of students: 2

Enter details for Student 1:
Enter Student ID: 1
Enter Name: ABC
Enter Marks for 5 subjects:
Bengali: 54
English: 200
You Enter the Wrong Value . Pls Enter The Mark <= 100 87
Computer Science : 89
Mathematics: 78
Physics: 68

Enter details for Student 2:
Enter Student ID: 2
Enter Name: Xyz
Enter Marks for 5 subjects:
Bengali: 46
English: 77
Computer Science : 99
Mathematics: 78
Physics: 65

Details of the student with the highest percentage:
Student ID: 1
Name: ABC
Marks for 5 subjects:
Bengali: 54
English: 87
Computer Science : 89
Mathematics: 78
Physics: 68
Percentage: 75.2%
~/Practical_Paper/00Pr_C++/As_4 B1:48

```

The Above Output We See That Our Program is work Well

Assignment 4

4.5 Program/Code Explanation :

```
#include <iostream>
using namespace std;
```

These lines include necessary header files for input/output operations (iostream) and declare that symbols from the std namespace will be used.

```
class Student {
private:
    int studId;
    string name;
    float marks[5], mark;
    float percentage;
    string sub[5]={"Bengali" , "English" , "Computer Science " , "Mathematics" , "Physices"};
```

- This defines a class named Student which represents student objects.
- It has private member variables studId, name, marks (an array to store marks of 5 subjects), mark (temporary variable to store each mark input), percentage, and sub (an array to store the names of 5 subjects).

```
void calculatePercentage() {
    float totalMarks = 0.0;
    for (int i = 0; i < 5; ++i) {
        totalMarks += marks[i];
    }
    percentage = totalMarks / 5.0;
}
```

- This is a private member function calculatePercentage() which calculates the percentage of marks for the student.
- It sums up the marks of all 5 subjects stored in the marks array and then calculates the percentage by dividing the total marks by 5.

```
void calculatePercentage() {
    float totalMarks = 0.0;
    for (int i = 0; i < 5; ++i) {
        totalMarks += marks[i];
    }
    percentage = totalMarks / 5.0;
}
```

- This is a private member function calculatePercentage() which calculates the percentage of marks for the student.
- It sums up the marks of all 5 subjects stored in the marks array and then calculates the percentage by dividing the total marks by 5.

```
public:
    void getData() {
        cout << "Enter Student ID: ";
        cin >> studId;
        cin.ignore();
        cout << "Enter Name: ";
        getline(cin, name);
        cout << "Enter Marks for 5 subjects:\n";
        for (int i = 0; i < 5; ++i) {
            cout << sub[i] << ": ";
            cin >> mark;
            cin.ignore();
            if (mark <= 100) {
                marks[i] = mark;
            } else {
                cout << "You Enter the Wrong Value. Please Enter The Mark <= 100: ";
                cin >> mark;
                marks[i] = mark;
            }
        }
        calculatePercentage();
    }
```

[illegible]

- ```
void show() const {
 cout << "Student ID: " << studId << endl;
 cout << "Name: " << name << endl;
 cout << "Marks for 5 subjects: \n";
 for (int i = 0; i < 5; ++i) {
 cout << sub[i] << ": " << marks[i] << endl;
 }
 cout << "Percentage: " << percentage << "%" << endl;
}
```

- ```
float getPercentage() const {
    return percentage;
}
```

```
int main() {
    int N;
    cout << "Enter the number of students: ";
    cin >> N;
    Student students[N];
    for (int i = 0; i < N; ++i) {
        cout << "\nEnter details for Student " << i + 1 << ":\n";
        students[i].getData();
    }
}
```

- ```
float highestPercentage = 0.0;
int indexHighestPercentage = 0;
for (int i = 0; i < N; ++i) {
 if (students[i].getPercentage() > highestPercentage) {
 highestPercentage = students[i].getPercentage();
 indexHighestPercentage = i;
 }
}
```

- ```
cout << "\nDetails of the student with the highest percentage:\n";
students[indexHighestPercentage].show();
return 0;
}
```



```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;

class Account {
protected:
    string customerName;
    int accountNumber;
    char accountType;
    float balance;
public:

    Account(const string &name, int accNumber, char accType, float initialBalance)
        : customerName(name), accountNumber(accNumber), accountType(accType), balance(initialBalance) {}

    void deposit(float amount) {
        balance += amount;
        cout << "Deposit Amount " << amount << " is Successful. Updated balance: " << balance << endl;
    }

    void displayBalance() const {
        cout << "Account Number: " << accountNumber << "\nBalance: " << balance << endl;
    }

    virtual void withdraw(float amount) = 0;

    virtual void checkMinimumBalance() = 0;
};

class SavingsAccount : public Account {
private:
    float interestRate;
public:

    SavingsAccount(const string &name, int accNumber, float initialBalance, float interest)
        : Account(name, accNumber, 'S', initialBalance), interestRate(interest) {}
};
```

[illegible]

1

Assignment 5

```

current.checkMinimumBalance();
current.displayBalance();

// Calculate interest for the savings account and update the balance
savings.calculateInterest();
savings.displayBalance();

return 0;
}

```

5.4 Program OutPut :

```

./ -- /list_Sem/Practical_Paper/OOPs_C++ 02:30 ls
./ -- / As_1/ AS_2/ AS_3/ AS_4/ AS_5/ 'MCA C++ Lab Assignments.pdf' .vscode/
./ -- /list_Sem/Practical_Paper/OOPs_C++ 02:30 cd AS_5/
./ -- /Practical_Paper/OOPs_C++/AS_5 02:30 ls
./ -- / bank* bank.cpp
./ -- /Practical_Paper/OOPs_C++/AS_5 02:30 ./bank
Deposit Amount 2000 is Successful. Updated balance: 7000
Account Number: 1038
Balance: 7000
Deposit Amount 3000 is Successful. Updated balance: 13000
Account Number: 13045
Balance: 13000
Withdrawal amount : 1000 successful. Updated balance: 6000
Account Number: 1038
Balance: 6000
Withdrawal amount 12000 successful. Updated balance: 1000
Account Number: 13045
Balance: 1000
Account Number: 1038
Balance: 6000
Minimum balance not maintained. Service charge :5 imposed. Updated balance: 995
Account Number: 13045
Balance: 995
Account Number: 1038
Balance: 6300
./ -- /Practical_Paper/OOPs_C++/AS_5 02:30

```

5.5 Program/Code Explanation :

```

#include <iostream>
#include <string>
#include <cmath>
using namespace std;

```

These lines include necessary header files for input/output operations (iostream), string manipulation (string), and mathematical functions (cmath).

```

class Account {
protected:
    string customerName;
    int accountNumber;
    char accountType;
    float balance;

public:
    Account(const string &name, int accNumber, char accType, float initialBalance)
        : customerName(name), accountNumber(accNumber), accountType(accType), balance(initialBalance) {}

```

- This defines a base class Account, which represents an account.
- It has protected member variables customerName, accountNumber, accountType, and balance.
- The class has a constructor that initializes these member variables.

```

void deposit(float amount) {
    balance += amount;
    cout << "Deposit Amount " << amount << " is Successful. Updated balance: " << balance << endl;
}

```

Assignment 5

- This member function deposit is used to deposit money into the account.
- It takes a parameter amount, adds it to the balance, and displays the updated balance.

```
void displayBalance() const {
    cout << "Account Number: " << accountNumber << "\nBalance: " << balance << endl;
}
```

- This member function displayBalance is used to display the account number and balance.

```
virtual void withdraw(float amount) = 0;
virtual void checkMinimumBalance() = 0;
```

- These are pure virtual functions, making the class abstract.
- The **withdraw** function is responsible for withdrawing money from the account.
- The **checkMinimumBalance** function is responsible for checking if the account has maintained the minimum balance.

```
class SavingsAccount : public Account {
private:
    float interestRate;

public:
    SavingsAccount(const string &name, int accNumber, float initialBalance, float interest)
        : Account(name, accNumber, 'S', initialBalance), interestRate(interest) {}
}
```

- This defines a derived class SavingsAccount, which inherits from the Account class.
- It adds an additional member variable interestRate to represent the interest rate for the savings account.
- It has a constructor that initializes the member variables of both the base and derived classes.

```
void calculateInterest() {
    float compoundInterest = balance * pow((1 + interestRate / 100), 1);
    balance = compoundInterest;
}
```

- This member function calculateInterest is used to calculate the interest for the savings account.
- It calculates the compound interest using the formula and updates the balance accordingly.

```
void withdraw(float amount) override {
    if (amount <= balance) {
        balance -= amount;
        cout << "Withdrawal amount : " << amount << " successful. Updated balance: " << balance << endl;
    } else {
        cout << "Insufficient funds for withdrawal." << endl;
    }
}

void checkMinimumBalance() override {
}
```

- These member functions withdraw and checkMinimumBalance are overrides of the virtual functions defined in the base class.
- withdraw function allows withdrawing money from the savings account, and it checks for sufficient funds.
- checkMinimumBalance function is not implemented for savings account

Assignment 5

```

class CurrentAccount : public Account {
private:
    float minimumBalance;
    float serviceCharge;

public:

    CurrentAccount(const string& name, int accNumber, float initialBalance, float minBalance, float charge)
        : Account(name, accNumber, 'C', initialBalance), minimumBalance(minBalance), serviceCharge(charge) {}

```

- This defines another derived class **CurrentAccount**, which also inherits from the **Account** class.
- It adds two additional member variables **minimumBalance** and **serviceCharge** to represent the minimum balance and service charge for the current account.
- It has a constructor that initializes the member variables of both the base and derived classes.

```

void withdraw(float amount) override {
    if (amount <= balance) {
        balance -= amount;
        std::cout << "Withdrawal amount " << amount << " successful. Updated balance: " << balance << endl;
    } else {
        cout << "Insufficient funds for withdrawal." << endl;
    }
}

void checkMinimumBalance() override {
    if (balance < minimumBalance) {
        balance -= serviceCharge;
        cout << "Minimum balance not maintained. Service charge :" << serviceCharge << " imposed. Updated
balance: " << balance << endl;
    }
}

```

- These member functions **withdraw** and **checkMinimumBalance** are overrides of the virtual functions defined in the base class.
- **withdraw** function allows withdrawing money from the current account, and it checks for sufficient funds.
- **checkMinimumBalance** function checks if the balance falls below the minimum balance and imposes a service charge if necessary.

```

int main() {
    // Create a SavingsAccount and a CurrentAccount
    SavingsAccount savings("Md Afsarul Hoque", 1038, 5000.0, 5.0); // 5% annual interest
    CurrentAccount current("Alfaj Hok", 13045, 10000.0, 2000.0, 5.0);

    // Deposit and display balance for both accounts
    savings.deposit(2000.0);
    savings.displayBalance();

    current.deposit(3000.0);
    current.displayBalance();

```

- This is the **main()** function, the entry point of the program.
- It creates instances of both **SavingsAccount** and **CurrentAccount** classes.
- It deposits money into both accounts and displays their balances.

```

// Withdraw and display balance for both accounts
savings.withdraw(1000.0);
savings.displayBalance();
// Insufficient funds for withdrawal
current.withdraw(12000.0);
current.displayBalance();

```

- It withdraws money from both accounts and displays their balances.
- It demonstrates that insufficient funds lead to a failure in withdrawal.

```

// Check minimum balance and impose penalty if necessary
savings.checkMinimumBalance();
savings.displayBalance();
current.checkMinimumBalance();
current.displayBalance();
// Calculate interest for the savings account and update the balance
savings.calculateInterest();
savings.displayBalance();
return 0;
}

```

- It checks if the minimum balance is maintained for both accounts and imposes a penalty if necessary.
- It displays the updated balances after imposing penalties.
- Finally, it calculates the interest for the savings account and updates the balance accordingly.
- It returns 0 to indicate successful execution.