# C++

Certainly! Here's the code for each of the questions you provided:

1. Write a program to print "Welcome to C++" using escape sequences: \n, \t, \r, \. Observe the difference in the output.

```
cppCopy code
#include <iostream>int main() {
    std::cout << "Welcome to C++" << std::endl;
    std::cout << "Welcome to C++\n";
    std::cout << "Welcome\tto\tC++\n";
    std::cout << "Welcome\rto C++\n";
    std::cout << "Welcome\\to C++\n";

    return 0;
}
```

1. Write a program to accept marks of five subjects from the user and calculate their average. Use implicit and explicit type conversion.

```
cppCopy code
#include <iostream>int main() {
    int marks1, marks2, marks3, marks4, marks5;

    std::cout << "Enter marks of five subjects: ";
    std::cin >> marks1 >> marks2 >> marks3 >> marks4 >> marks5;

    // Implicit type conversion
    int total1 = marks1 + marks2 + marks3 + marks4 + marks5;
    double average1 = total1 / 5.0;

    // Explicit type conversion
    int total2 = static_cast<int>(marks1) + static_cast<int>(marks2) + static_cast<int>(ma
rks3) + static_cast<int>(marks4) + static_cast<int>(marks5);
    double average2 = static_cast<double>(total2) / 5;

    std::cout << "Average (implicit): " << average1 << std::endl;
    std::cout << "Average (explicit): " << average2 << std::endl;

    return 0;
}
```

1. Write a program to add two numbers and store the result in a third variable. Print the result.

```
cppCopy code
#include <iostream>int main() {
    int num1, num2, sum;

    std::cout << "Enter two numbers: ";
    std::cin >> num1 >> num2;

    sum = num1 + num2;

    std::cout << "Sum: " << sum << std::endl;

    return 0;
}
```

1. Write a program to swap two variables using a third variable and without using a third variable.

```
cppCopy code
#include <iostream>int main() {
    int num1, num2;

    std::cout << "Enter two numbers: ";
    std::cin >> num1 >> num2;

    std::cout << "Before swapping: num1 = " << num1 << ", num2 = " << num2 << std::endl;

    // Using a third variable
    int temp = num1;
    num1 = num2;
    num2 = temp;

    std::cout << "After swapping (using a third variable): num1 = " << num1 << ", num2 = "
<< num2 << std::endl;

    // Without using a third variable
    num1 = num1 + num2;
    num2 = num1 - num2;
    num1 = num1 - num2;

    std::cout << "After swapping (without using a third variable): num1 = " << num1 << ",
 num2 = " << num2 << std::endl;

    return 0;
}
```

1. Write a program to calculate Net Salary of an employee. Accept Basic Salary (BS) from the user.
   HRA is 15% of BS
   DA is 30% of BS
   PF is 12.5% of GS (Gross Salary is BS + HRA + DA)
   Net Salary = Gross Salary - PF

```
cppCopy code
#include <iostream>int main() {
    double basicSalary, hra, da, pf, grossSalary, netSalary;

    std::cout << "Enter Basic Salary: ";
    std::cin >> basicSalary;

    hra = 0.15 * basicSalary;
    da = 0.30 * basicSalary;
    grossSalary = basicSalary + hra + da;
    pf = 0.125 * grossSalary;
    netSalary = grossSalary - pf;

    std::cout << "HRA: " << hra << std::endl;
    std::cout << "DA: " << da << std::endl;
    std::cout << "Gross Salary: " << grossSalary << std::endl;
    std::cout << "PF: " << pf << std::endl;
    std::cout << "Net Salary: " << netSalary << std::endl;

    return 0;
}
```

1. Write a program to find a leap year using:
   a. if-else and logical operators (&& and ||)

```
cppCopy code
#include <iostream>int main() {
    int year;

    std::cout << "Enter a year: ";
    std::cin >> year;

    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
        std::cout << year << " is a leap year." << std::endl;
    } else {
        std::cout << year << " is not a leap year." << std::endl;
    }

    return 0;
}
```

b. Conditional Operators (? :)

```
cppCopy code
#include <iostream>int main() {
    int year;

    std::cout << "Enter a year: ";
    std::cin >> year;

    std::string result = ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) ? "is a l
eap year." : "is not a leap year.";

    std::cout << year << " " << result << std::endl;

    return 0;
}
```

1. Write a program to accept the basic salary and total sales amount for a salesperson and calculate commission according to the sales amount. Display the net salary and commission earned.

```
cppCopy code
#include <iostream>int main() {
    double basicSalary, totalSales, commission, netSalary;
    std::cout << "Enter Basic Salary: ";
    std::cin >> basicSalary;

    std::cout << "Enter Total Sales Amount: ";
    std::cin >> totalSales;

    if (totalSales >= 5000 && totalSales <= 7500) {
        commission = 0.03 * totalSales;
    } else if (totalSales > 7500 && totalSales <= 10500) {
        commission = 0.08 * totalSales;
    } else if (totalSales > 10500 && totalSales <= 15000) {
        commission = 0.11 * totalSales;
    } else if (totalSales > 15000) {
        commission = 0.15 * totalSales;
    } else {
        commission = 0;
    }

    netSalary = basicSalary + commission;

    std::cout << "Commission: " << commission << std::endl;
    std::cout << "Net Salary: " << netSalary << std::endl;

    return 0;
}
```

1. Using the switch-case construct, write a menu-driven program to perform basic calculations (addition, subtraction, multiplication, and division) on two user-entered numbers.

```
cppCopy code
#include <iostream>int main() {
    int choice, num1, num2;

    std::cout << "Enter two numbers: ";
    std::cin >> num1 >> num2;

    std::cout << "Menu:\n";
    std::cout << "1. Addition\n";
    std::cout << "2. Subtraction\n";
    std::cout << "3. Multiplication\n";
    std::cout << "4. Division\n";
    std::cout << "Enter your choice: ";
    std::cin >> choice;

    switch (choice) {
        case 1:
            std::cout << "Addition: " << num1 + num2 << std::endl;
            break;
        case 2:
            std::cout << "Subtraction: " << num1 - num2 << std::endl;
            break;
        case 3:
            std::cout << "Multiplication: " << num1 * num2 << std::endl;
            break;
        case 4:
            if (num2 != 0) {
                std::cout << "Division: " << static_cast<double>(num1) / num2 << std::end
l;
            } else {
                std::cout << "Error: Division by zero is not allowed." << std::endl;
            }
            break;
        default:
            std::cout << "Invalid choice!" << std::endl;
    }

    return 0;
}
```

1. Write a program to find the maximum of three numbers using conditional operators.

```
cppCopy code
#include <iostream>int main() {
    int num1, num2, num3;

    std::cout << "Enter three numbers: ";
    std::cin >> num1 >> num2 >> num3;

    int max = (num1 > num2) ? ((num1 > num3) ? num1 : num3) : ((num2 > num3) ? num2 : num
3);

    std::cout << "Maximum: " << max << std::endl;

    return 0;
}
```

1. Write a program to convert the user-entered character into its opposite case. The program should also display an error message if the character entered by the user is not an alphabet.

```
cppCopy code
#include <iostream>int main() {
    char ch;

    std::cout << "Enter a character: ";
    std::cin >> ch;

    if (isalpha(ch)) {
        if (islower(ch)) {
            ch = toupper(ch);
        } else {
            ch = tolower(ch);
        }

        std::cout << "Opposite Case: " << ch << std::endl;
    } else {
        std::cout << "Error: Not an alphabet!" << std::endl;
    }

    return 0;
}
```

Certainly! Here's the code for each of the questions you provided:

1. Write a program to print whether the user-entered number is an Armstrong number. An Armstrong number is one for which the sum of the cube of all its digits is the same as the number.

```
cppCopy code
#include <iostream>int main() {
    int number, originalNumber, remainder, result = 0;

    std::cout << "Enter a number: ";
    std::cin >> number;

    originalNumber = number;

    while (originalNumber != 0) {
        remainder = originalNumber % 10;
        result += remainder * remainder * remainder;
        originalNumber /= 10;
    }

    if (result == number) {
        std::cout << number << " is an Armstrong number." << std::endl;
    } else {
        std::cout << number << " is not an Armstrong number." << std::endl;
    }

    return 0;
}
```

1. Write a program to display whether a user-entered number is prime or not.

```
cppCopy code
#include <iostream>bool isPrime(int number) {
    if (number <= 1) {
        return false;
    }

    for (int i = 2; i <= number / 2; ++i) {
        if (number % i == 0) {
            return false;
        }
    }

    return true;
}

int main() {
    int number;

    std::cout << "Enter a number: ";
    std::cin >> number;

    if (isPrime(number)) {
        std::cout << number << " is a prime number." << std::endl;
    } else {
        std::cout << number << " is not a prime number." << std::endl;
    }

    return 0;
}
```

1. Write a program to display the first 'n' prime numbers in sequence, where 'n' is accepted from the user.

```
cppCopy code
#include <iostream>bool isPrime(int number) {
    if (number <= 1) {
        return false;
    }

    for (int i = 2; i <= number / 2; ++i) {
        if (number % i == 0) {
            return false;
        }
    }
    return true;
}

int main() {
    int n;

    std::cout << "Enter the value of 'n': ";
    std::cin >> n;

    std::cout << "The first " << n << " prime numbers are: ";

    int count = 0;
    int number = 2;

    while (count < n) {
        if (isPrime(number)) {
            std::cout << number << " ";
            ++count;
        }

        ++number;
    }

    std::cout << std::endl;

    return 0;
}
```

1. Write a program to generate all possible combinations of 1, 2, 3 using a for loop.

```
cppCopy code
#include <iostream>int main() {
    std::cout << "All possible combinations of 1, 2, 3 are: " << std::endl;

    for (int i = 1; i <= 3; ++i) {
        for (int j = 1; j <= 3; ++j) {
            for (int k = 1; k <= 3; ++k) {
                std::cout << i << " " << j << " " << k << std::endl;
            }
        }
    }

    return 0;
}
```

1. Write a program for a matchstick game between the computer and the user. The program should ensure that the computer always wins. The rules for the game are as follows:
   a. There are 21 matchsticks.
   b. The computer asks the player to pick 1, 2, 3, or 4 matchsticks.
   c. The player is given the chance to pick the sticks first, then the computer picks the sticks.
   d. Whoever is forced to pick up the last matchstick loses the game.

```
cppCopy code
#include <iostream>int main() {
    int matchsticks = 21;
    int playerPick, computerPick;

    while (true) {
        std::cout << "Number of matchsticks available: " << matchsticks << std::endl;
        std::cout << "Pick 1, 2, 3, or 4 matchsticks: ";
        std::cin >> playerPick;

        if (playerPick < 1 || playerPick > 4 || playerPick > matchsticks) {
            std::cout << "Invalid input! Pick again." << std::endl;
            continue;
        }

        matchsticks -= playerPick;

        if (matchsticks <= 1) {
            std::cout << "You picked the last matchstick. You lose!" << std::endl;
            break;
        }

        computerPick = 5 - playerPick;

        std::cout << "Computer picks " << computerPick << " matchsticks." << std::endl;

        matchsticks -= computerPick;

        if (matchsticks <= 1) {
            std::cout << "Computer picked the last matchstick. You win!" << std::endl;
            break;
        }
    }

    return 0;
}
```

1. Write a program to generate the following output:

```
Copy code
1 2 3 4 5 4 3 2 1
1 2 3 4   4 3 2 1
1 2 3      3 2 1
1 2           2 1
1                1
```

```
cppCopy code
#include <iostream>int main() {
    for (int i = 1; i <= 5; ++i) {
        for (int j = 1; j <= i; ++j) {
            std::cout << j << " ";
        }

        for (int k = 5 - i; k > 0; --k) {
            std::cout << "  ";
        }

        for (int l = i; l > 0; --l) {
            std::cout << l << " ";
        }

        std::cout << std::endl;
    }

    return 0;
}
```

I hope the provided code helps! Let me know if you have any further questions.

Q26) •  Write a program that calculates the average marks of all the subjects. The number of subjects ‘n’ is accepted from the user.  Allocate memory dynamically for ‘n’ integers.  Free the memory when not in use.

Code -

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

int main() {
int n;
```

```
cout << "Enter the number of subjects: ";
cin >> n;
```

```
// Dynamically allocate memory for n integers
int* marks = new int[n];

// Accept marks for each subject
cout << "Enter marks for " << n << " subjects:" << endl;
for (int i = 0; i < n; i++) {
    cout << "Subject " << i + 1 << ": ";
    cin >> marks[i];
}

// Calculate the sum of marks
int sum = 0;
for (int i = 0; i < n; i++) {
    sum += marks[i];
}

// Calculate the average
double average = static_cast<double>(sum) / n;

// Display the average marks
cout << "Average marks: " << average << endl;

// Free the dynamically allocated memory
delete[] marks;

return 0;
```

```
}
```

Q27) Write a program to sort five user entered strings into alphabetical order. Use dynamic memory allocation to store strings.

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

void sortStrings(char** strings, int size) {
bool sorted = false;
while (!sorted) {
sorted = true;
for (int i = 0; i < size - 1; i++) {
if (strcmp(strings[i], strings[i + 1]) > 0) {
```

```
swap(strings[i], strings[i + 1]);
sorted = false;
}
}
}
}
```

```
int main() {
const int size = 5;
char** strings = new char*[size];
```

```
// Accept user input for strings
for (int i = 0;#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter the number of subjects: ";
    cin >> n;

    // Dynamically allocate memory for n integers
    int* marks = new int[n];

    // Accept marks for each subject
    cout << "Enter marks for " << n << " subjects:" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Subject " << i + 1 << ": ";
        cin >> marks[i];
    }

    // Calculate the sum of marks
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += marks[i];
    }

    // Calculate the average
    double average = static_cast<double>(sum) / n;

    // Display the average marks
    cout << "Average marks: " << average << endl;

    // Free the dynamically allocated memory
    delete[] marks;

    return 0;
} i < size; i++) {
    cout << "Enter string " << i + 1 << ": ";
    char buffer[100];
    cin.getline(buffer, 100);

    // Allocate memory for the string
    strings[i] = new char[strlen(buffer) + 1];
    strcpy(strings[i], buffer);
}

// Sort the strings
sortStrings(strings, size);

// Display the sorted strings
cout << "Sorted strings: " << endl;
for (int i = 0; i < size; i++) {
    cout << strings[i] << endl;
}

// Deallocate memory
for (int i = 0; i < size; i++) {
    delete[] strings[i];
}
delete[] strings;

return 0;
```

}

## Functions – Call by Value

17 Write a function ‘Fibonacci()’ to generate the terms of the fibonacci series. The number of terms should be entered by the user in main(). The series should be printed in the function. The terms of the Fibonacci series are 0 1 1 2 3 5 8 13 21 . .

ChatGPT

Certainly! Here's the code for the Fibonacci series generator function:

```
cppCopy code
#include <iostream>void Fibonacci(int n) {
    int first = 0;
    int second = 1;

    std::cout << "Fibonacci series with " << n << " terms: ";

    for (int i = 0; i < n; ++i) {
        std::cout << first << " ";

        int next = first + second;
        first = second;
        second = next;
    }

    std::cout << std::endl;
}

int main() {
    int numTerms;

    std::cout << "Enter the number of terms for the Fibonacci series: ";
    std::cin >> numTerms;

    Fibonacci(numTerms);

    return 0;
}
```

In this code, the function `Fibonacci()` takes the number of terms as a parameter and generates the Fibonacci series using a loop. The series is then printed within the function. The `main()` function prompts the user to enter the number of terms and calls the `Fibonacci()` function to generate and display the series.

I hope this helps! Let me know if you have any further questions.

Question 18a: Write a single function `areaCircum()` that calculates the area and circumference of a circle. The values of area and circumference should be printed in `main()`.

```
cppCopy code
#include <iostream>void areaCircum(float radius, float& area, float& circumference) {
    const float PI = 3.14159;
    area = PI * radius * radius;
    circumference = 2 * PI * radius;
}

int main() {
    float radius;
    float area, circumference;

    std::cout << "Enter the radius of the circle: ";
    std::cin >> radius;

    areaCircum(radius, area, circumference);

    std::cout << "Area of the circle: " << area << std::endl;
    std::cout << "Circumference of the circle: " << circumference << std::endl;

    return 0;
}
```

This program prompts the user to enter the radius of a circle. It then calls the `areaCircum()` function to calculate the area and circumference of the circle using the provided radius. Finally, it prints the calculated values in the `main()` function.

Let me know if you have any questions or if you would like to move on to the next question.

solve question line by line Functions – Call by Address

18 Write a menu driven program, which allows the user to select either a circle or rectangle.
a Write a single function areaCircum() that calculates the area and circumference of a circle. The values of area and circumference should be printed in main()
b Write a single function areaPeri() that calculates the area and perimeter of a rectangle. The values of area and perimeter should be printed in main()
19 Write a program which accepts two integers in main(). Pass these as arguments to function max(). In max(), find the maximum of these two numbers and using a return statement, return the address of the maximum number and print it in main().

ChatGPT

Sure! Let's solve each part of the question step by step.

Question 18a: Write a single function `areaCircum()` that calculates the area and circumference of a circle. The values of area and circumference should be printed in `main()`.

```
cppCopy code
#include <iostream>void areaCircum(float radius, float& area, float& circumference) {
    const float PI = 3.14159;
    area = PI * radius * radius;
    circumference = 2 * PI * radius;
}

int main() {
    float radius;
    float area, circumference;

    std::cout << "Enter the radius of the circle: ";
    std::cin >> radius;

    areaCircum(radius, area, circumference);

    std::cout << "Area of the circle: " << area << std::endl;
    std::cout << "Circumference of the circle: " << circumference << std::endl;

    return 0;
}
```

This program prompts the user to enter the radius of a circle. It then calls the `areaCircum()` function to calculate the area and circumference of the circle using the provided radius. Finally, it prints the calculated values in the `main()` function.

Please let me know if you would like to proceed with the next part or if you have any questions regarding this solution.

next question

ChatGPT

Question 18b: Write a single function `areaPeri()` that calculates the area and perimeter of a rectangle. The values of area and perimeter should be printed in `main()`.

```
cppCopy code
#include <iostream>void areaPeri(float length, float width, float& area, float& perimeter)
{
    area = length * width;
    perimeter = 2 * (length + width);
}

int main() {
    float length, width;
    float area, perimeter;

    std::cout << "Enter the length of the rectangle: ";
    std::cin >> length;

    std::cout << "Enter the width of the rectangle: ";
    std::cin >> width;

    areaPeri(length, width, area, perimeter);

    std::cout << "Area of the rectangle: " << area << std::endl;
    std::cout << "Perimeter of the rectangle: " << perimeter << std::endl;

    return 0;
}
```

This program prompts the user to enter the length and width of a rectangle. It then calls the `areaPeri()` function to calculate the area and perimeter of the rectangle using the provided length and width. Finally, it prints the calculated values in the `main()` function.

Please let me know if you would like to proceed with the next question or if you have any further questions regarding this solution.

next

ChatGPT

Question 19: Write a program that accepts two integers in `main()`. Pass these as arguments to the function `max()`. In `max()`, find the maximum of these two numbers and using a return statement, return the address of the maximum number and print it in `main()`.

```
cppCopy code
#include <iostream>int* max(int* num1, int* num2) {
    if (*num1 > *num2) {
        return num1;
    } else {
        return num2;
    }
}

int main() {
    int num1, num2;

    std::cout << "Enter the first number: ";
    std::cin >> num1;

    std::cout << "Enter the second number: ";
    std::cin >> num2;

    int* maxNum = max(&num1, &num2);

    std::cout << "The maximum number is: " << *maxNum << std::endl;

    return 0;
}
```

In this program, the `max()` function takes two integer pointers as arguments and compares the values they point to. It returns the address of the maximum number using a return statement. In the `main()` function, the user is prompted to enter two numbers, which are stored in `num1` and `num2`. The `max()` function is then called with the addresses of these variables, and the returned address is stored in the `maxNum` pointer. Finally, the value at the address pointed to by `maxNum` is printed as the maximum number.

Please let me know if there is anything else I can help you with.

20.a) Accept five integers in an array and find the maximum and minimum of the integers. Do not sort the array.

```
cppCopy code
#include <iostream>void findMaxMin(int arr[], int size, int& maxNum, int& minNum) {
    maxNum = arr[0];
    minNum = arr[0];

    for (int i = 1; i < size; i++) {
        if (arr[i] > maxNum) {
            maxNum = arr[i];
        }
        if (arr[i] < minNum) {
            minNum = arr[i];
        }
    }
}

int main() {
    const int SIZE = 5;
    int arr[SIZE];

    std::cout << "Enter five integers:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        std::cin >> arr[i];
    }

    int maxNum, minNum;
    findMaxMin(arr, SIZE, maxNum, minNum);

    std::cout << "Maximum number: " << maxNum << std::endl;
    std::cout << "Minimum number: " << minNum << std::endl;

    return 0;
}
```

In this program, the function `findMaxMin()` accepts an array of integers, its size, and two integer references for the maximum and minimum numbers. It initializes the `maxNum` and `minNum` variables with the first element of the array and then iterates over the remaining elements to update the maximum and minimum values accordingly. In the `main()` function, the user is prompted to enter five integers, which are stored in the `arr` array. The `findMaxMin()` function is called with the array, its size, and the `maxNum` and `minNum` variables passed by reference. Finally, the maximum and minimum numbers are displayed.

20.b) Multiply each element of the array by 5 and store it in another array and display it.

```
cppCopy code
#include <iostream>void multiplyBy5(int arr[], int size, int result[]) {
    for (int i = 0; i < size; i++) {
        result[i] = arr[i] * 5;
    }
}

int main() {
    const int SIZE = 5;
    int arr[SIZE];
    int result[SIZE];

    std::cout << "Enter five integers:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        std::cin >> arr[i];
    }

    multiplyBy5(arr, SIZE, result);

    std::cout << "Resultant array after multiplying by 5:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        std::cout << result[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

In this program, the function `multiplyBy5()` accepts an array of integers, its size, and another array to store the multiplied values. It multiplies each element of the input array by 5 and stores the result in the `result` array. In the `main()` function, the user is prompted to enter five integers, which are stored in the `arr` array. The `multiplyBy5()` function is called with the `arr` array, its size, and the `result` array. Finally, the `result` array is displayed, showing the elements after multiplying by 5.

Please let me know if there is anything else I can help you with.

next question

ChatGPT

1. Write a program to delete an element from a user-entered array. Accept the position of the element to be deleted from the user.

```
cppCopy code
#include <iostream>void deleteElement(int arr[], int& size, int position) {
    if (position >= 0 && position < size) {
        for (int i = position; i < size - 1; i++) {
            arr[i] = arr[i + 1];
        }
        size--;
        std::cout << "Element deleted successfully." << std::endl;
    } else {
        std::cout << "Invalid position. Deletion failed." << std::endl;
    }
}

int main() {
    const int MAX_SIZE = 100;
    int arr[MAX_SIZE];
    int size;

    std::cout << "Enter the size of the array: ";
    std::cin >> size;
    std::cout << "Enter " << size << " elements: ";
    for (int i = 0; i < size; i++) {
        std::cin >> arr[i];
    }

    int position;
    std::cout << "Enter the position of the element to delete: ";
    std::cin >> position;

    deleteElement(arr, size, position);

    std::cout << "Array after deletion: ";
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

In this program, the function `deleteElement()` accepts an array of integers, its size, and the position of the element to be deleted. It first checks if the position is within the valid range of the array. If so, it shifts all the elements after the specified position to the left, effectively removing the desired element. The `size` variable is decremented to reflect the new size of the array. If the position is invalid, an error message is displayed.

In the `main()` function, the user is prompted to enter the size of the array and the elements. Then, the position of the element to delete is entered. The `deleteElement()` function is called with the array, its size, and the position. Finally, the updated array after deletion is displayed.

Please let me know if there is anything else I can help you with.

next

ChatGPT

1. Write a program to insert an element into an array. Accept the position of the element to be inserted from the user.

```
cppCopy code
#include <iostream>void insertElement(int arr[], int& size, int position, int element) {
    if (position >= 0 && position <= size) {
        for (int i = size - 1; i >= position; i--) {
            arr[i + 1] = arr[i];
        }
        arr[position] = element;
        size++;
        std::cout << "Element inserted successfully." << std::endl;
    } else {
        std::cout << "Invalid position. Insertion failed." << std::endl;
    }
}

int main() {
    const int MAX_SIZE = 100;
    int arr[MAX_SIZE];
    int size;

    std::cout << "Enter the size of the array: ";
    std::cin >> size;

    std::cout << "Enter " << size << " elements: ";
    for (int i = 0; i < size; i++) {
        std::cin >> arr[i];
    }

    int position, element;
    std::cout << "Enter the position to insert the element: ";
    std::cin >> position;
    std::cout << "Enter the element to insert: ";
    std::cin >> element;

    insertElement(arr, size, position, element);

    std::cout << "Array after insertion: ";
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

In this program, the function `insertElement()` accepts an array of integers, its size, the position where the element should be inserted, and the element itself. It first checks if the position is within the valid range of the array (including the end of the array). If so, it shifts all the elements from the specified position to the right to make room for the new element. Then, it assigns the new element to the desired position and increments the `size` variable to reflect the updated size of the array. If the position is invalid, an error message is displayed.

In the `main()` function, the user is prompted to enter the size of the array and the elements. Then, the position and the element to insert are entered. The `insertElement()` function is called with the array, its size, the position, and the element. Finally, the updated array after insertion is displayed.

Please let me know if there is anything else I can assist you with.

next

ChatGPT

1. Write a program to accept and display a 3x3 matrix. Write `Accept()` and `Display()` functions to perform the tasks.
   a. Find the transpose of the matrix and print the transpose using the `Display()` function.
   b. Accept another matrix of the same dimensions. Find the addition of the two matrices and print the resultant matrix.

```
cppCopy code
#include <iostream>const int SIZE = 3;

void Accept(int matrix[][SIZE]) {
    std::cout << "Enter the elements of the matrix:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            std::cout << "Enter element at position (" << i << ", " << j << "): ";
            std::cin >> matrix[i][j];
        }
    }
}

void Display(int matrix[][SIZE]) {
    std::cout << "Matrix:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }
}

void Transpose(int matrix[][SIZE], int result[][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            result[j][i] = matrix[i][j];
        }
    }
}

void AddMatrices(int matrix1[][SIZE], int matrix2[][SIZE], int result[][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            result[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }
}

int main() {
    int matrix1[SIZE][SIZE];
    int matrix2[SIZE][SIZE];
    int transpose[SIZE][SIZE];
    int sum[SIZE][SIZE];

    std::cout << "Enter the elements for the first matrix:" << std::endl;
    Accept(matrix1);

    std::cout << "Enter the elements for the second matrix:" << std::endl;
    Accept(matrix2);

    std::cout << std::endl;

    std::cout << "Displaying the first matrix:" << std::endl;
    Display(matrix1);

    std::cout << std::endl;

    std::cout << "Displaying the second matrix:" << std::endl;
    Display(matrix2);

    std::cout << std::endl;

    Transpose(matrix1, transpose);
    std::cout << "Transpose of the first matrix:" << std::endl;
    Display(transpose);

    std::cout << std::endl;

    AddMatrices(matrix1, matrix2, sum);
    std::cout << "Sum of the two matrices:" << std::endl;
    Display(sum);

    return 0;
```

```
}
```

In this program, the `Accept()` function is used to accept the elements of a 3x3 matrix from the user. It takes a 2D array as a parameter and uses nested loops to iterate over the matrix elements, prompting the user to enter each element.

The `Display()` function is used to display the matrix. It also takes a 2D array as a parameter and uses nested loops to iterate over the matrix elements, printing each element.

The `Transpose()` function calculates the transpose of a matrix. It takes the input matrix and a result matrix as parameters. It uses nested loops to swap the elements of the

next

ChatGPT

1. Write a program which will calculate the multiplication of two 3x3 matrices.

```
cppCopy code
#include <iostream>const int SIZE = 3;

void MultiplyMatrices(int matrix1[][SIZE], int matrix2[][SIZE], int result[][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            result[i][j] = 0;
            for (int k = 0; k < SIZE; k++) {
                result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }
}

void Display(int matrix[][SIZE]) {
    std::cout << "Matrix:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            std::cout << matrix[i][j] << " ";
        }
        std::cout << std::endl;
    }
}
```

```
int main() {
    int matrix1[SIZE][SIZE];
    int matrix2[SIZE][SIZE];
    int product[SIZE][SIZE];

    std::cout << "Enter the elements for the first matrix:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            std::cout << "Enter element at position (" << i << ", " << j << "): ";
            std::cin >> matrix1[i][j];
        }
    }

    std::cout << "Enter the elements for the second matrix:" << std::endl;
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            std::cout << "Enter element at position (" << i << ", " << j << "): ";
            std::cin >> matrix2[i][j];
        }
    }

    MultiplyMatrices(matrix1, matrix2, product);

    std::cout << "Product of the two matrices:" << std::endl;
    Display(product);

    return 0;
}
```

In this program, the `MultiplyMatrices()` function calculates the multiplication of two 3x3 matrices. It takes the input matrices and a result matrix as parameters. It uses nested loops to iterate over the matrices and performs the matrix multiplication algorithm to calculate the product.

The `Display()` function is used to display the matrix. It takes a 2D array as a parameter and uses nested loops to iterate over the matrix elements, printing each element.

In the `main()` function, the user is prompted to enter the elements for both matrices. The matrices are then passed to the `MultiplyMatrices()` function to calculate the product. Finally, the product matrix is displayed using the `Display()` function.

next

ChatGPT

1. Write a program to accept a string from the user. Write separate functions to perform the following:
   a. Find the number of occurrences of a given character in the string.
   b. Find the number of blank spaces in the string.
   c. Find the number of words in the string.
   d. Find the total number of all the vowels in the string.

```
cppCopy code
#include <iostream>#include <string>#include <cctype>int CountOccurrences(const std::string& str, char ch) {
    int count = 0;
    for (char c : str) {
        if (c == ch) {
            count++;
        }
    }
    return count;
}

int CountBlankSpaces(const std::string& str) {
    int count = 0;
    for (char c : str) {
        if (c == ' ') {
            count++;
        }
    }
    return count;
}

int CountWords(const std::string& str) {
    int count = 0;
    bool inWord = false;
    for (char c : str) {
        if (std::isalpha(c)) {
            if (!inWord) {
                count++;
                inWord = true;
            }
        } else {
            inWord = false;
        }
    }
    return count;
}

int CountVowels(const std::string& str) {
    int count = 0;
    for (char c : str) {
        char lowercaseChar = std::tolower(c);
        if (lowercaseChar == 'a' || lowercaseChar == 'e' || lowercaseChar == 'i' || lowerc
aseChar == 'o' || lowercaseChar == 'u') {
            count++;
        }
    }
    return count;
}

int main() {
    std::string str;

    std::cout << "Enter a string: ";
    std::getline(std::cin, str);

    char ch;
    std::cout << "Enter a character to count its occurrences: ";
    std::cin >> ch;

    int occurrences = CountOccurrences(str, ch);
    std::cout << "Number of occurrences of '" << ch << "' in the string: " << occurrences
 << std::endl;

    int blankSpaces = CountBlankSpaces(str);
    std::cout << "Number of blank spaces in the string: " << blankSpaces << std::endl;

    int words = CountWords(str);
    std::cout << "Number of words in the string: " << words << std::endl;

    int vowels = CountVowels(str);
    std::cout << "Number of vowels in the string: " << vowels << std::endl;

    return 0;
}
```

In this program, separate functions are defined to perform different operations on a given string:

- `CountOccurrences()` takes a string and a character as parameters and counts the number of occurrences of that character in the string.
- `CountBlankSpaces()` counts the number of blank spaces in the string.
- `CountWords()` counts the number of words in the string. It uses a flag `inWord` to keep track of whether the current character is part of a word or not.

- `CountVowels()` counts the number of vowels (case-insensitive) in the string. It converts each character to lowercase using `std::tolower()` and checks if it is a vowel.

In the `main()` function, the user is prompted to enter a string. Then, the user can enter a character to count its occurrences, and the program calls the respective functions to perform the operations. The results are displayed on the console.

next

ChatGPT

1. Write a program that calculates the average marks of all the subjects. The number of subjects 'n' is accepted from the user. Allocate memory dynamically for 'n' integers. Free the memory when not in use.

```
cppCopy code
#include <iostream>double calculateAverageMarks(int* marks, int numSubjects) {
    int sum = 0;
    for (int i = 0; i < numSubjects; i++) {
        sum += marks[i];
    }
    return static_cast<double>(sum) / numSubjects;
}

int main() {
    int numSubjects;
    std::cout << "Enter the number of subjects: ";
    std::cin >> numSubjects;

    int* marks = new int[numSubjects];

    std::cout << "Enter the marks for each subject:" << std::endl;
    for (int i = 0; i < numSubjects; i++) {
        std::cout << "Subject " << (i + 1) << ": ";
        std::cin >> marks[i];
    }

    double averageMarks = calculateAverageMarks(marks, numSubjects);
    std::cout << "Average marks: " << averageMarks << std::endl;

    delete[] marks;

    return 0;
}
```

In this program, the user is prompted to enter the number of subjects. Then, dynamic memory allocation is performed using `new` to allocate an array of integers of size `numSubjects` to store the marks.

The user is then prompted to enter the marks for each subject, and the marks are stored in the dynamically allocated array.

The function `calculateAverageMarks()` takes the array of marks and the number of subjects as parameters. It calculates the sum of all the marks and returns the average by dividing the sum by the number of subjects.

Finally, the average marks are displayed on the console, and the dynamically allocated memory is freed using `delete[]` to release the memory when it is no longer needed.

Note: It is important to free the dynamically allocated memory using `delete[]` to avoid memory leaks.

next

ChatGPT

1. Write a program to sort five user-entered strings into alphabetical order. Use dynamic memory allocation to store strings.

```
cppCopy code
#include <iostream>#include <string>void sortStrings(std::string* strings, int numStrings)
{
    for (int i = 0; i < numStrings - 1; i++) {
        for (int j = i + 1; j < numStrings; j++) {
            if (strings[i] > strings[j]) {
                std::swap(strings[i], strings[j]);
            }
        }
    }
}

int main() {
    const int numStrings = 5;
    std::string* strings = new std::string[numStrings];
    std::cout << "Enter " << numStrings << " strings:" << std::endl;
    for (int i = 0; i < numStrings; i++) {
        std::cout << "String " << (i + 1) << ": ";
        std::cin >> strings[i];
    }

    sortStrings(strings, numStrings);

    std::cout << "Sorted strings:" << std::endl;
    for (int i = 0; i < numStrings; i++) {
        std::cout << strings[i] << std::endl;
    }

    delete[] strings;

    return 0;
}
```

In this program, dynamic memory allocation is used to store an array of strings. The user is prompted to enter five strings, and each string is stored in the dynamically allocated array.

The function `sortStrings()` takes the array of strings and the number of strings as parameters. It uses a simple bubble sort algorithm to sort the strings in alphabetical order.

After sorting the strings, the program displays the sorted strings on the console.

Finally, the dynamically allocated memory is freed using `delete[]` to release the memory when it is no longer needed.

Note: It's important to free the dynamically allocated memory using `delete[]` to avoid memory leaks.

Structures

28 Define a structure “Book” having members – bookId, title, price.  Use typedef to name this structure “BOOK”.  Accept the data for a book and display the record.

29 Modify above assignment - 28 to hold records of five books.  Display the records of all the books using a function.

30 Write a program to copy one structure into another –

a On element by element basis.
b Copying an entire structure to another.

ChatGPT

1. Here's an implementation of the structure "Book" using typedef:

```
cppCopy code
#include <iostream>#include <string>typedef struct {
    int bookId;
    std::string title;
    float price;
} BOOK;

int main() {
    BOOK book;

    std::cout << "Enter book ID: ";
    std::cin >> book.bookId;

    std::cout << "Enter book title: ";
    std::cin.ignore();
    std::getline(std::cin, book.title);

    std::cout << "Enter book price: ";
    std::cin >> book.price;

    std::cout << "\nBook Details:\n";
    std::cout << "Book ID: " << book.bookId << std::endl;
    std::cout << "Title: " << book.title << std::endl;
    std::cout << "Price: " << book.price << std::endl;

    return 0;
}
```

In the above code, a structure named "Book" is defined using typedef as "BOOK". The structure has three members: `bookId` of type `int`, `title` of type `std::string`, and `price` of type `float`. The `BOOK` typedef allows us to use `BOOK` as a shorthand for the structure type.

The program accepts the data for a book from the user and displays the entered record using the `std::cout` statements.

1. Here's a modified implementation to hold records of five books and display them using a function:

```
cppCopy code
#include <iostream>#include <string>typedef struct {
    int bookId;
    std::string title;
    float price;
} BOOK;

void displayBooks(const BOOK books[], int numBooks) {
    std::cout << "\nBook Records:\n";
    for (int i = 0; i < numBooks; i++) {
        std::cout << "Book " << i + 1 << ":\n";
        std::cout << "Book ID: " << books[i].bookId << std::endl;
        std::cout << "Title: " << books[i].title << std::endl;
        std::cout << "Price: " << books[i].price << std::endl;
        std::cout << std::endl;
    }
}

int main() {
    const int numBooks = 5;
    BOOK books[numBooks];

    for (int i = 0; i < numBooks; i++) {
        std::cout << "Enter details for Book " << i + 1 << ":\n";
        std::cout << "Enter book ID: ";
        std::cin >> books[i].bookId;

        std::cout << "Enter book title: ";
        std::cin.ignore();
        std::getline(std::cin, books[i].title);

        std::cout << "Enter book price: ";
        std::cin >> books[i].price;

        std::cout << std::endl;
    }

    displayBooks(books, numBooks);

    return 0;
}
```

In this code, an array of `BOOK` structures is created to hold records of five books. The `displayBooks` function takes an array of `BOOK` structures and the number of books as arguments. It iterates through the array and displays the details of each book using the `std::cout` statements.

The main function prompts the user to enter the details of each book and stores them in the array of `BOOK` structures. After accepting all the records, it calls the `displayBooks` function to display the records of all the books.

1. Here's a program to copy one structure into another on an element by element basis and by copying the entire structure:

```
cppCopy code
#include <iostream>#include <string>typedef struct {
    int bookId;
    std::string title;
    float price;
} BOOK;

void copyByElements(const BOOK& source, BOOK& destination) {
    destination.bookId = source.bookId;
    destination.title = source.title;
    destination.price = source.price;
}

void copyByStructure(const BOOK& source, BOOK& destination) {
    destination = source;
}

int main() {
    BOOK book1 = { 1, "Book 1", 10.99 };
    BOOK book2;
    BOOK book3;

    // Copy by elements
    copyByElements(book1, book2);

    // Copy by structure
    copyByStructure(book1, book3);

    std::cout << "Copied Book 1 to Book 2:\n";
    std::cout << "Book 2 ID: " << book2.bookId << std::endl;
    std::cout << "Book 2 Title: " << book2.title << std::endl;
    std::cout << "Book 2 Price: " << book2.price << std::endl;

    std::cout << "\nCopied Book 1 to Book 3:\n";
    std::cout << "Book 3 ID: " << book3.bookId << std::endl;
    std::cout << "Book 3 Title: " << book3.title << std::endl;
    std::cout << "Book 3 Price: " << book3.price << std::endl;

    return 0;
}
```

In this code, a structure named "Book" is defined using typedef. It has three members: `bookId` of type `int`, `title` of type `std::string`, and `price` of type `float`.

The `copyByElements` function takes the source structure and the destination structure as references. It copies each member of the source structure into the corresponding member of the destination structure.

The `copyByStructure` function takes the source structure and the destination structure as references. It copies the entire source structure into the destination structure using the assignment operator.

In the main function, two additional structures `book2` and `book3` are created. The `copyByElements` function is used to copy the values from `book1` to `book2` on an element by element basis. The `copyByStructure` function is used to copy the entire `book1` structure to `book3`.

Finally, the values of `book2` and `book3` are displayed using `std::cout` statements to verify the copying process.

Let me know if you have any further questions!

Q31)

- Implement class Complex with real (int) and imaginary (int) parts as data members of class. Implement below functions inside this class.

default constructor, parameterized constructor, getter functions for real and imaginary parts, setter functions for real and imaginary parts, display function.

code -

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

class Complex {
int real;
int imaginary;

public:
// Default constructor
Complex() {
real = 0;
```

imaginary = 0;

}

```
// Parameterized constructor
Complex(int r, int i) {
    real = r;
    imaginary = i;
}

// Getter functions
int getReal() {
    return real;
}

int getImaginary() {
    return imaginary;
}

// Setter functions
void setReal(int r) {
    real = r;
}

void setImaginary(int i) {
    imaginary = i;
}

// Display function
void display() {
    cout << real << " + " << imaginary << "i" << endl;
}
```

};

int main() {

Complex c1; // Default constructor

Complex c2(3, 4); // Parameterized constructor

```
// Displaying the complex numbers
cout << "c1: ";
c1.display();

cout << "c2: ";
c2.display();

// Using getter functions
cout << "Real part of c2: " << c2.getReal() << endl;
cout << "Imaginary part of c2: " << c2.getImaginary() << endl;
```

```
// Using setter functions
c1.setReal(1);
c1.setImaginary(2);
cout << "c1 after setting values: ";
c1.display();

return 0;
```

}

Implement class Complex with real (int) and imaginary (int) parts as data members of class. Implement below functions inside this class.
default constructor, parameterized constructor, getter functions for real and imaginary parts, setter functions for real and imaginary parts, display function.

32 Write a program to count number of objects created for above class Complex . Use static data member and member function.

ChatGPT

1. Implementation of the `Complex` class:

```
cppCopy code
#include <iostream>class Complex {
private:
    int real;
    int imaginary;

public:
    // Default constructor
    Complex() {
        real = 0;
        imaginary = 0;
    }

    // Parameterized constructor
    Complex(int r, int i) {
        real = r;
        imaginary = i;
    }

    // Getter function for real part
    int getReal() {
        return real;
    }

    // Getter function for imaginary part
    int getImaginary() {
        return imaginary;
    }

    // Setter function for real part
    void setReal(int r) {
        real = r;
    }

    // Setter function for imaginary part
    void setImaginary(int i) {
        imaginary = i;
    }

    // Display function to print complex number
    void display() {
        std::cout << real << " + " << imaginary << "i" << std::endl;
    }
};
```

In the above code, the `Complex` class is implemented with the required member functions.

The default constructor initializes the `real` and `imaginary` parts to 0.

The parameterized constructor takes two integers as arguments and assigns them to the `real` and `imaginary` parts.

The getter functions `getReal()` and `getImaginary()` return the values of the `real` and `imaginary` parts, respectively.

The setter functions `setReal()` and `setImaginary()` set the values of the `real` and `imaginary` parts, respectively.

The `display()` function prints the complex number in the format `real + imaginaryi`.

1. Counting the number of objects created for the `Complex` class using static data member and member function:

```
cppCopy code
#include <iostream>class Complex {
private:
    int real;
    int imaginary;
    static int count; // static data member to count objects

public:
    Complex() {
        real = 0;
        imaginary = 0;
        count++; // increment count when an object is created
    }

    // ...

    static int getCount() { // static member function to get the count
        return count;
    }
};

int Complex::count = 0; // initialization of static data member

int main() {
    Complex c1;
    Complex c2;
    Complex c3;

    std::cout << "Number of objects created: " << Complex::getCount() << std::endl;

    return 0;
}
```

In the above code, a static data member `count` is added to the `Complex` class to keep track of the number of objects created.

The `count` variable is initialized outside the class definition.

Each time a new object of the `Complex` class is created, the `count` variable is incremented in the default constructor.

To access the static member `count`, the static member function `getCount()` is defined, which returns the value of `count`.

In the `main()` function, three objects of the `Complex` class are created, and then the number of objects created is displayed by calling the `getCount()` function using the class name.

next

ChatGPT

1. Implementation of the class `Array`:

```
cppCopy code
#include <iostream>class Array {
private:
    int size;
    int* arr;

public:
    // Default constructor
    Array() {
        size = 5;
        arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = 0;
        }
    }

    // Parameterized constructor
    Array(int s) {
        size = s;
        arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = 0;
        }
    }

    // Copy constructor
    Array(const Array& other) {
        size = other.size;
        arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = other.arr[i];
        }
    }

    // Destructor
    ~Array() {
        delete[] arr;
    }
};
```

In the above code, the `Array` class is implemented with the required constructors and destructor.

The default constructor initializes the `size` member variable to 5 and dynamically allocates an array of integers with size 5. It also initializes all the elements of the array to 0.

The parameterized constructor takes an integer `s` as an argument, assigns it to the `size` member variable, and dynamically allocates an array of integers with size `s`. It also initializes all the elements of the array to 0.

The copy constructor takes a constant reference to another `Array` object as an argument. It performs a deep copy of the array by creating a new array with the same size and copying the elements from the other object's array to the newly created array.

The destructor deallocates the dynamically allocated memory for the array using the `delete[]` operator.

Note: This implementation does not include the implementation of other member functions like getter/setter functions or display functions. You can add those functions as per your requirements.

Let me know if you have any further questions!

Q32) • Write a program to count number of objects created for above class Complex . Use static data member and member function.

#include <iostream>
using namespace std;

class Complex {
double real;
double imaginary;
static int count; // Static data member to count objects

public:
Complex(double r = 0, double i = 0) {
real = r;
imaginary = i;
count++; // Increment count when an object is created
}

```
static int getCount() {
    return count; // Return the count of objects
}
```

};

int Complex::count = 0; // Initialize the static data member

int main() {
Complex c1(3, 4);
Complex c2(1, 2);
Complex c3(5, 6);

```
cout << "Number of objects created: " << Complex::getCount() << endl;

return 0;
```

}

Q33)

- Write a class Array which will store integer elements in dynamically allocated space.

class Array

{

int size;

int *arr;

}

Provide below functions in Array class -

1. Default constructor - Take 5 as default size. Initialize the array elements to 0
2. Parameterized constructor - Takes size as parameter. Initialize the array elements to 0.
3. Copy constructor - Performs deep copy of Array object.

4. Destructor - Deletes memory which is allocated dynamically

code -

#include <iostream>
using namespace std;

```
class Array {
int size;
int* arr;

public:
// Default constructor
Array() {
size = 5;
arr = new int[size];
for (int i = 0; i < size; i++) {
arr[i] = 0;
}
}
```

```
// Parameterized constructor
Array(int newSize) {
    size = newSize;
    arr = new int[size];
    for (int i = 0; i < size; i++) {
        arr[i] = 0;
    }
}

// Copy constructor
Array(const Array& other) {
    size = other.size;
    arr = new int[size];
    for (int i = 0; i < size; i++) {
        arr[i] = other.arr[i];
    }
}

// Destructor
~Array() {
    delete[] arr;
}
```

```
};

int main() {
// Create an object using the default constructor
Array arr1;
```

```
// Create an object using the parameterized constructor
Array arr2(10);

// Create an object using the copy constructor
Array arr3(arr1);

return 0;
```

```
}
```

Q34)

- Write a class Employee for an application which will have data members for employee id, employee name and salary. Provide the following functionalities in Employee class.

1. Initialing objects using default and parameterized constructors.
2. Accepting and displaying the information of employee from console

```
class Employee
{
int emp_id;
char emp_name[20];
double salary;
public:
Employee();
Employee(int, char*, double);
void Accept();
void Display();
};
```

In "main" function declare an array of Employee objects. Write a menu driven program to -

1. Insert record into an array.
2. Update information of specific employee on the basis of emp_id accepted from user
3. Display all records.

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

class Employee {
int emp_id;
char emp_name[20];
double salary;

public:
Employee() {
emp_id = 0;
strcpy(emp_name, "");
salary = 0.0;
}
```

```
Employee(int id, const char* name, double sal) {
    emp_id = id;
    strcpy(emp_name, name);
    salary = sal;
}

void Accept() {
    cout << "Enter Employee ID: ";
    cin >> emp_id;
    cin.ignore(); // Ignore the newline character in the input buffer
    cout << "Enter Employee Name: ";
    cin.getline(emp_name, 20);
    cout << "Enter Salary: ";
    cin >> salary;
}

void Display() {
    cout << "Employee ID: " << emp_id << endl;
    cout << "Employee Name: " << emp_name << endl;
    cout << "Salary: " << salary << endl;
}

int getEmployeeId() const {
    return emp_id;
}

void setEmployeeId(int id) {
    emp_id = id;
}
```

};

int main() {

const int MAX_EMPLOYEES = 100;

Employee employees[MAX_EMPLOYEES];

int employeeCount = 0;

```
int choice;
do {
    cout << "Menu:\\n"
        << "1. Insert Record\\n"
        << "2. Update Employee Information\\n"
        << "3. Display All Records\\n"
        << "4. Exit\\n"
        << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
    case 1:
        if (employeeCount >= MAX_EMPLOYEES) {
            cout << "Maximum limit reached. Cannot insert more records." << endl;
            break;
        }
        employees[employeeCount].Accept();
        employeeCount++;
        cout << "Record inserted successfully." << endl;
        break;
    case 2: {
        int empId;
        bool found = false;

        cout << "Enter Employee ID to update information: ";
        cin >> empId;

        for (int i = 0; i < employeeCount; i++) {
            if (employees[i].getEmployeeId() == empId) {
                employees[i].Accept();
                found = true;
                cout << "Information updated successfully." << endl;
                break;
            }
        }
        if (!found) {
            cout << "Employee with ID " << empId << " not found." << endl;
        }

        break;
    }
    case 3:
        if (employeeCount == 0) {
            cout << "No records available." << endl;
            break;
        }
        cout << "Employee Records:" << endl;
        for (int i = 0; i < employeeCount; i++) {
            employees[i].Display();
            cout << endl;
        }
        break;
    case 4:
        cout << "Exiting program." << endl;
        break;
    default:
        cout << "Invalid choice. Please try again." << endl;
    }

    cout << endl;
} while (choice != 4);

return 0;
```

}

@Q35

- Implement the class Vehicle_Policy, whose data members are PolicyId, Vehicle_Name and Amount. Create default and parameterized constructors, getters/setter functions for each attribute. Implement a menu driven program to perform below operations –
    - Accept valid data from the user and store it in an array of Vehicle_Policy objects. Display the stored data.
    - Sort data in decreasing order of policy amount.
    - Display all the details for the holder with highest and lowest policy amount.

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

class Vehicle_Policy {
private:
int PolicyId;
string Vehicle_Name;
double Amount;

public:
Vehicle_Policy() {
PolicyId = 0;
Amount = 0.0;
}
```

```
Vehicle_Policy(int policyId, const string& vehicleName, double amount) {
    PolicyId = policyId;
    Vehicle_Name = vehicleName;
    Amount = amount;
}

int getPolicyId() const {
    return PolicyId;
}

void setPolicyId(int policyId) {
    PolicyId = policyId;
}

string getVehicleName() const {
    return Vehicle_Name;
}

void setVehicleName(const string& vehicleName) {
    Vehicle_Name = vehicleName;
}

double getAmount() const {
    return Amount;
}

void setAmount(double amount) {
    Amount = amount;
}
```

```
};

void displayPolicies(const Vehicle_Policy policies[], int count) {
cout << "Policy Details:" << endl;
for (int i = 0; i < count; i++) {
cout << "Policy ID: " << policies[i].getPolicyId() << endl;
cout << "Vehicle Name: " << policies[i].getVehicleName() << endl;
cout << "Policy Amount: " << policies[i].getAmount() << endl;
cout << endl;
}
}

bool compareAmount(const Vehicle_Policy& policy1, const Vehicle_Policy& policy2) {
return policy1.getAmount() > policy2.getAmount();
}

void sortPolicies(Vehicle_Policy policies[], int count) {
sort(policies, policies + count, compareAmount);
}

void displayHighestAndLowest(const Vehicle_Policy policies[], int count) {
if (count == 0) {
cout << "No policies available." << endl;
return;
}
```

```
cout << "Policy with the highest amount:" << endl;
cout << "Policy ID: " << policies[0].getPolicyId() << endl;
cout << "Vehicle Name: " << policies[0].getVehicleName() << endl;
cout << "Policy Amount: " << policies[0].getAmount() << endl;

cout << "Policy with the lowest amount:" << endl;
cout << "Policy ID: " << policies[count - 1].getPolicyId() << endl;
cout << "Vehicle Name: " << policies[count - 1].getVehicleName() << endl;
cout << "Policy Amount: " << policies[count - 1].getAmount() << endl;
```

```
}

int main() {
const int MAX_POLICIES = 100;
```

Vehicle_Policy policies[MAX_POLICIES];
int policyCount = 0;

```
int choice;
do {
    cout << "Menu:\\n"
        << "1. Add Policy\\n"
        << "2. Display Policies\\n"
        << "3. Sort Policies by Amount\\n"
        << "4. Display Highest and Lowest Policy\\n"
        << "5. Exit\\n"
        << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
    case 1: {
        int policyId;
        string vehicleName;
        double amount;

        cout << "Enter Policy ID: ";
        cin >> policyId;
        cout << "Enter Vehicle Name: ";
        cin.ignore();
        getline(cin, vehicleName);
        cout << "Enter Policy Amount: ";
        cin >> amount;

        policies[policyCount] = Vehicle_Policy(policyId, vehicleName, amount);
        policyCount++;
        cout << "Policy added successfully." << endl;
        break;
    }
    case 2:
        displayPolicies(policies, policyCount);
        break;
    case 3:
        sortPolicies(policies, policyCount);
        cout << "Policies sorted by amount in decreasing order." << endl;
        break;
    case 4:
        displayHighestAndLowest(policies, policyCount);
        break;
    case 5:
        cout << "Exiting program." << endl;
        break;
    default:
        cout << "Invalid choice. Please try again." << endl;
    }

    cout << endl;
} while (choice != 5);

return 0;
```

}

question 33 and 37

- Modify class Array in assignment – 33 to overload below operators

1. Overloaded [] operator - used to get/set array element e.g. arr[4] gives 4 index element

   th

2. Overloaded ! operator - reverses array elements.
3. Overload >> - Accepts array size from user. Also accepts elements of array from user.
4. Overload << - Displays array.

#include <iostream>
using namespace std;

class Array {
private:
int size;
int* arr;

public:
Array() {
size = 5;
arr = new int[size];
initializeArray();
}

```
Array(int n) {
    size = n;
    arr = new int[size];
    initializeArray();
}

Array(const Array& other) {
    size = other.size;
    arr = new int[size];
    for (int i = 0; i < size; i++) {
        arr[i] = other.arr[i];
    }
}

~Array() {
    delete[] arr;
}

int& operator[](int index) {
    if (index >= 0 && index < size)
        return arr[index];
    else
        throw out_of_range("Index out of range");
}

Array& operator!() {
    int start = 0;
    int end = size - 1;
    while (start < end) {
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;
        start++;
        end--;
    }
    return *this;
}

friend istream& operator>>(istream& in, Array& a) {
    cout << "Enter elements of the array:" << endl;
    for (int i = 0; i < a.size; i++) {
        in >> a.arr[i];
    }
    return in;
}

friend ostream& operator<<(ostream& out, const Array& a) {
    out << "Array elements: ";
    for (int i = 0; i < a.size; i++) {
        out << a.arr[i] << " ";
    }
    out << endl;
    return out;
}
```

private:

void initializeArray() {

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

```
arr[i] = 0;
}
}
};
```

q38

- Write a class Matrix which will store elements of integer matrix (two dimensional array) in dynamically allocated space. Provide below functions in matrix class -

1. Default constructor - Take 3 as default row and column size. Initialize the matrix elements to 0
2. Parameterized constructor - Takes row and column size as parameters. Initialize matrix to 0.
3. Copy constructor - Performs deep copy of matrix object.
4. Destructor - Deletes memory which is allocated dynamically.
5. Overloaded + operator - Performs addition of two matrices.
6. Overloaded ! operator - Calculates transpose of a user entered square matrix.
7. Overload >> - Accepts row & column size from user. Also accepts matrix elements from user.
8. Overload << - Displays matrix.

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

class Matrix {
private:
int rows;
int columns;
int** mat;

public:
Matrix(int r = 3, int c = 3) : rows(r), columns(c) {
mat = new int*[rows];
for (int i = 0; i < rows; i++) {
mat[i] = new int[columns]{};
}
}
```

```
Matrix(const Matrix& other) : rows(other.rows), columns(other.columns) {
    mat = new int*[rows];
    for (int i = 0; i < rows; i++) {
        mat[i] = new int[columns]{};
        for (int j = 0; j < columns; j++) {
            mat[i][j] = other.mat[i][j];
        }
    }
}

~Matrix() {
    for (int i = 0; i < rows; i++) {
        delete[] mat[i];
    }
    delete[] mat;
}

Matrix operator+(const Matrix& other) {
    if (rows != other.rows || columns != other.columns) {
        throw invalid_argument("Matrices must have the same dimensions for addition");
    }

    Matrix result(rows, columns);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            result.mat[i][j] = mat[i][j] + other.mat[i][j];
        }
    }
    return result;
}

Matrix operator!() {
    if (rows != columns) {
        throw invalid_argument("Matrix must be square for transpose");
    }

    Matrix result(rows, columns);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            result.mat[i][j] = mat[j][i];
        }
    }
    return result;
}
```

```
friend istream& operator>>(istream& in, Matrix& m) {
    cout << "Enter matrix elements:" << endl;
    for (int i = 0; i < m.rows; i++) {
        for (int j = 0; j < m.columns; j++) {
            in >> m.mat[i][j];
        }
    }
    return in;
}

friend ostream& operator<<(ostream& out, const Matrix& m) {
    out << "Matrix elements:" << endl;
    for (int i = 0; i < m.rows; i++) {
        for (int j = 0; j < m.columns; j++) {
            out << m.mat[i][j] << " ";
        }
        out << endl;
    }
    return out;
}
```

```
};
```

q39

- Write a class Student having following –
    - Student Roll Number (int)
    - Student Name (char array)
    - Date of Birth (Date class object where Date is user defined class)

Implement default constructor, parameterized constructor, accept, display. Generate the student roll number automatically.

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

class Date {
private:
int day;
int month;
int year;
```

```
public:
Date(int d = 1, int m = 1, int y = 2000) : day(d), month(m), year(y) {}
```

```
void display() const {
    cout << day << "/" << month << "/" << year;
}
```

```
};
```

```
class Student {
private:
int rollNumber;
char name[50];
Date dateOfBirth;
```

```
static int nextRollNumber;
```

```
public:
Student() : rollNumber(nextRollNumber++) {
name[0] = '\0';
}
```

```
Student(const char* studentName, const Date& dob) : rollNumber(nextRollNumber++), dateOfBi
rth(dob) {
    strncpy(name, studentName, sizeof(name) - 1);
    name[sizeof(name) - 1] = '\\0';
}

void accept() {
    cout << "Enter student name: ";
    cin.getline(name, sizeof(name));

    int day, month, year;
    cout << "Enter date of birth (dd mm yyyy): ";
    cin >> day >> month >> year;
    cin.ignore(); // Ignore the newline character after entering the year

    dateOfBirth = Date(day, month, year);
}

void display() const {
    cout << "Roll Number: " << rollNumber << endl;
    cout << "Name: " << name << endl;
    cout << "Date of Birth: ";
    dateOfBirth.display();
    cout << endl;
}
```

};

int Student::nextRollNumber = 1;

int main() {
Student student1;
student1.accept();

```
Student student2("John Doe", Date(10, 5, 2002));

cout << "\\nStudent 1:\\n";
student1.display();

cout << "\\nStudent 2:\\n";
student2.display();

return 0;
```

}

40)

- Construct a hierarchy of employees.

1. Create an Employee class with attributes like employee id, name, date of birth.
2. Inherit class WageEmployee from base class Employee
3. WageEmployee class should have following members
4. Number of hours worked
5. Rate per hour
6. SalesPerson class should have following members
7. Number of items sold
8. Commission per item

9. Write constructors for WageEmployee and SalesPerson classes.
10. Override the functions for displaying details, calculating salary in WageEmployee and SalesPerson class.

WageEmployee Salary = hours * rate

SalesPerson Salary = hours*rate + sales*commission

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

class Employee {
protected:
int employeeId;
string name;
string dateOfBirth;

public:
Employee(int id, const string& empName, const string& dob)
: employeeId(id), name(empName), dateOfBirth(dob) {}
```

```
virtual void displayDetails() const {
    cout << "Employee ID: " << employeeId << endl;
    cout << "Name: " << name << endl;
    cout << "Date of Birth: " << dateOfBirth << endl;
}

virtual double calculateSalary() const = 0;
```

```
};

class WageEmployee : public Employee {
private:
double hoursWorked;
double ratePerHour;

public:
WageEmployee(int id, const string& empName, const string& dob,
```

double hours, double rate)
: Employee(id, empName, dob), hoursWorked(hours), ratePerHour(rate) {}

```
void displayDetails() const override {
    cout << "-- Wage Employee --" << endl;
    Employee::displayDetails();
    cout << "Hours Worked: " << hoursWorked << endl;
    cout << "Rate per Hour: " << ratePerHour << endl;
}

double calculateSalary() const override {
    return hoursWorked * ratePerHour;
}
```

};

class SalesPerson : public Employee {
private:
int itemsSold;
double commissionPerItem;

public:
SalesPerson(int id, const string& empName, const string& dob,
int items, double commission)
: Employee(id, empName, dob), itemsSold(items), commissionPerItem(commission) {}

```
void displayDetails() const override {
    cout << "-- Sales Person --" << endl;
    Employee::displayDetails();
    cout << "Items Sold: " << itemsSold << endl;
    cout << "Commission per Item: " << commissionPerItem << endl;
}

double calculateSalary() const override {
    return itemsSold * commissionPerItem;
}
```

};

int main() {
Employee* employee1 = new WageEmployee(1, "John Doe", "01/01/1990", 40, 15.0);
Employee* employee2 = new SalesPerson(2, "Jane Smith", "02/02/1995", 100, 2.5);

```
employee1->displayDetails();
cout << "Salary: $" << employee1->calculateSalary() << endl;

cout << endl;

employee2->displayDetails();
cout << "Salary: $" << employee2->calculateSalary() << endl;

delete employee1;
delete employee2;

return 0;
```

}

Q41)

- Construct a hierarchy of employees using classes Employee, Manager and MarketingExecutive.

Create an Employee class with attributes - EmpId, EmpName and BasicSal

Write default and parameterized constructors for Employee class.

Inherit class Manager and MarketingExecutive from base class Employee. Implement

necessary constructors in derived classes.

Manager class should have members for -

1. Petrol Allowance (double): 7% of BasicSal
2. Food Allowance (double): 11% of BasicSal
3. Other Allowance(double): 4% of BasicSal

MarketingExecutive class should have members for -

1. Kilometers Travelled
2. Tour Allowance(integer): Rs.5/- per kilometer
3. Telephone Allowance(double): Rs.2000/-

Override the functions for accepting/displaying details, calculating gross and net salary in Manager and MarketingExecutive class.

Gross Salary = BasicSal + Allowances

Net Salary = Gross Salary - PF

PF = 12.5% of BasicSal

Create valid objects for all the classes. Accept and display details for those objects.

Print net and gross salary for those objects details.

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

class Employee {
protected:
int empId;
string empName;
double basicSal;

public:
Employee() : empId(0), basicSal(0.0) {}
```

```
Employee(int id, const string& name, double sal)
    : empId(id), empName(name), basicSal(sal) {}

void acceptDetails() {
    cout << "Enter Employee ID: ";
    cin >> empId;
    cin.ignore();

    cout << "Enter Employee Name: ";
    getline(cin, empName);

    cout << "Enter Basic Salary: ";
    cin >> basicSal;
}

void displayDetails() const {
    cout << "Employee ID: " << empId << endl;
    cout << "Employee Name: " << empName << endl;
    cout << "Basic Salary: " << basicSal << endl;
}

virtual double calculateGrossSalary() const = 0;

double calculateNetSalary() const {
    double pf = 0.125 * basicSal;
    return calculateGrossSalary() - pf;
}
```

```
};
```

```
class Manager : public Employee {
private:
double petrolAllowance;
double foodAllowance;
double otherAllowance;
```

```
public:
Manager() : petrolAllowance(0.07), foodAllowance(0.11), otherAllowance(0.04) {}
```

```
Manager(int id, const string& name, double sal)
    : Employee(id, name, sal), petrolAllowance(0.07 * sal), foodAllowance(0.11 * sal), otherAllowance(0.04 * sal) {}

void acceptDetails() {
    Employee::acceptDetails();
}

void displayDetails() const {
    cout << "-- Manager Details --" << endl;
    Employee::displayDetails();
    cout << "Petrol Allowance: " << petrolAllowance << endl;
    cout << "Food Allowance: " << foodAllowance << endl;
    cout << "Other Allowance: " << otherAllowance << endl;
}

double calculateGrossSalary() const {
    return basicSal + petrolAllowance + foodAllowance + otherAllowance;
}
```

```
};
```

```
class MarketingExecutive : public Employee {
private:
int kilometersTravelled;
int tourAllowance;
double telephoneAllowance;
```

```
public:
MarketingExecutive() : tourAllowance(5), telephoneAllowance(2000.0) {}
MarketingExecutive(int id, const string& name, double sal)
    : Employee(id, name, sal), tourAllowance(5), telephoneAllowance(2000.0) {}

void acceptDetails() {
    Employee::acceptDetails();

    cout << "Enter Kilometers Travelled: ";
    cin >> kilometersTravelled;
}

void displayDetails() const {
    cout << "-- Marketing Executive Details --" << endl;
    Employee::displayDetails();
    cout << "Kilometers Travelled: " << kilometersTravelled << endl;
    cout << "Tour Allowance: " << (tourAllowance * kilometersTravelled) << endl;
    cout << "Telephone Allowance: " << telephoneAllowance << endl;
}

double calculateGrossSalary() const {
    return basicSal + (tourAllowance * kilometersTravelled) + telephoneAllowance;
}
```

};

int main() {
Manager manager;
manager.acceptDetails();

```
MarketingExecutive executive;
executive.acceptDetails();

cout << "\\n-- Manager Details --" << endl;
manager.displayDetails();
cout << "Gross Salary: " << manager.calculateGrossSalary() << endl;
cout << "Net Salary: " << manager.calculateNetSalary() << endl;

cout << "\\n-- Marketing Executive Details --" << endl;
executive.displayDetails();
cout << "Gross Salary: " << executive.calculateGrossSalary() << endl;
cout << "Net Salary: " << executive.calculateNetSalary() << endl;

return 0;
```

}

- Modify above assignment 40 as given below –
    - Create Employee pointer and allow it to hold address of any Employee objects (WageEmployee & SalesPerson) created on heap
    - Using Employee pointer call accept() and display() to accept and display the details of dynamically created objects.
    - Free the memory for the objects created on heap

Make the Employee class  as abstract

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

class Employee {
protected:
int empId;
string empName;
double basicSal;

public:
Employee() : empId(0), basicSal(0.0) {}
```

```
Employee(int id, const string& name, double sal)
    : empId(id), empName(name), basicSal(sal) {}

virtual void accept() = 0;
virtual void display() const = 0;
virtual double calculateGrossSalary() const = 0;

double calculateNetSalary() const {
    double pf = 0.125 * basicSal;
    return calculateGrossSalary() - pf;
}

virtual ~Employee() {}
```

```
};

class WageEmployee : public Employee {
private:
```

int hoursWorked;
double ratePerHour;

public:
WageEmployee() : hoursWorked(0), ratePerHour(0.0) {}

```
WageEmployee(int id, const string& name, double sal, int hours, double rate)
    : Employee(id, name, sal), hoursWorked(hours), ratePerHour(rate) {}

void accept() {
    cout << "Enter Employee ID: ";
    cin >> empId;
    cin.ignore();

    cout << "Enter Employee Name: ";
    getline(cin, empName);

    cout << "Enter Basic Salary: ";
    cin >> basicSal;

    cout << "Enter Hours Worked: ";
    cin >> hoursWorked;

    cout << "Enter Rate per Hour: ";
    cin >> ratePerHour;
}

void display() const {
    cout << "-- Wage Employee Details --" << endl;
    cout << "Employee ID: " << empId << endl;
    cout << "Employee Name: " << empName << endl;
    cout << "Basic Salary: " << basicSal << endl;
    cout << "Hours Worked: " << hoursWorked << endl;
    cout << "Rate per Hour: " << ratePerHour << endl;
    cout << "Gross Salary: " << calculateGrossSalary() << endl;
    cout << "Net Salary: " << calculateNetSalary() << endl;
}

double calculateGrossSalary() const {
    return hoursWorked * ratePerHour;
}
```

};

class SalesPerson : public Employee {
private:
int itemsSold;
double commissionPerItem;

public:

SalesPerson() : itemsSold(0), commissionPerItem(0.0) {}

```
SalesPerson(int id, const string& name, double sal, int sold, double commission)
    : Employee(id, name, sal), itemsSold(sold), commissionPerItem(commission) {}

void accept() {
    cout << "Enter Employee ID: ";
    cin >> empId;
    cin.ignore();

    cout << "Enter Employee Name: ";
    getline(cin, empName);

    cout << "Enter Basic Salary: ";
    cin >> basicSal;

    cout << "Enter Items Sold: ";
    cin >> itemsSold;

    cout << "Enter Commission per Item: ";
    cin >> commissionPerItem;
}

void display() const {
    cout << "-- Sales Person Details --" << endl;
    cout << "Employee ID: " << empId << endl;
    cout << "Employee Name: " << empName << endl;
    cout << "Basic Salary: " << basicSal << endl;
    cout << "Items Sold: " << itemsSold << endl;
    cout << "Commission per Item: " << commissionPerItem << endl;
    cout << "Gross Salary: " << calculateGrossSalary() << endl;
    cout << "Net Salary: " << calculateNetSalary() << endl;
}

double calculateGrossSalary() const {
    return basicSal + (itemsSold * commissionPerItem);
}
```

};

int main() {

Employee* empPtr;

```
// Creating WageEmployee object dynamically
empPtr = new WageEmployee();
empPtr->accept();
empPtr->display();
```

```
delete empPtr; // Freeing the memory

cout << endl;

// Creating SalesPerson object dynamically
empPtr = new SalesPerson();
empPtr->accept();
empPtr->display();

delete empPtr; // Freeing the memory

return 0;
```

}

- Extend the assignment 40 as per below requirements.
    - Derive class Manager publicly from class Employee with data members for basic salary(int) and number of sub-ordinates(int).
    - Implement necessary functions inside class Manager
    - Derive class SalesManager publicly from SalesPerson and Manager. SalesManager do not have any of its own data members.
    - Implements necessary functions inside class SalesManager.
-

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

class Employee {
protected:
int empId;
string empName;
string dob;

public:
Employee() : empId(0), empName(""), dob("") {}

```
Employee(int id, const string& name, const string& dateOfBirth)
    : empId(id), empName(name), dob(dateOfBirth) {}

virtual void display() const {
    cout << "Employee ID: " << empId << endl;
    cout << "Employee Name: " << empName << endl;
    cout << "Date of Birth: " << dob << endl;
}
```

};

class WageEmployee : public Employee {
private:
int hoursWorked;
double ratePerHour;

public:
WageEmployee() : Employee(), hoursWorked(0), ratePerHour(0.0) {}

```
WageEmployee(int id, const string& name, const string& dateOfBirth, int hours, double rat
e)
    : Employee(id, name, dateOfBirth), hoursWorked(hours), ratePerHour(rate) {}

void display() const {
    cout << "-- Wage Employee Details --" << endl;
    Employee::display();
    cout << "Hours Worked: " << hoursWorked << endl;
    cout << "Rate per Hour: " << ratePerHour << endl;
    cout << "Salary: " << calculateSalary() << endl;
}

double calculateSalary() const {
    return hoursWorked * ratePerHour;
}
```

};

class SalesPerson : public Employee {
private:
int itemsSold;
double commissionPerItem;

public:
SalesPerson() : Employee(), itemsSold(0), commissionPerItem(0.0) {}

```
SalesPerson(int id, const string& name, const string& dateOfBirth, int sold, double commis
sion)
    : Employee(id, name, dateOfBirth), itemsSold(sold), commissionPerItem(commission) {}

void display() const {
    cout << "-- Sales Person Details --" << endl;
    Employee::display();
    cout << "Items Sold: " << itemsSold << endl;
    cout << "Commission per Item: " << commissionPerItem << endl;
    cout << "Salary: " << calculateSalary() << endl;
}

double calculateSalary() const {
    return (hoursWorked * ratePerHour) + (itemsSold * commissionPerItem);
}
```

};

class Manager : public Employee {
private:
int basicSalary;
int numOfSubordinates;

public:
Manager() : Employee(), basicSalary(0), numOfSubordinates(0) {}

```
Manager(int id, const string& name, const string& dateOfBirth, int salary, int subordinate
s)
    : Employee(id, name, dateOfBirth), basicSalary(salary), numOfSubordinates(subordinate
s) {}

void display() const {
    cout << "-- Manager Details --" << endl;
    Employee::display();
    cout << "Basic Salary: " << basicSalary << endl;
    cout << "Number of Subordinates: " << numOfSubordinates << endl;
    cout << "Salary: " << calculateSalary() << endl;
}

double calculateSalary() const {
    return basicSalary;
}
```

};

class SalesManager : public SalesPerson, public Manager {
public:
SalesManager() : SalesPerson(), Manager() {}

```
SalesManager(int id, const string& name, const string& dateOfBirth, int sold, double commission, int salary, int subordinates)
    : SalesPerson(id, name, dateOfBirth, sold, commission), Manager(id, name, dateOfBirth, salary, subordinates) {}

void display() const {
    cout << "-- Sales Manager Details --" << endl;
    Employee::display();
    cout << "Items Sold: " << itemsSold << endl;
    cout << "Commission per Item: " << commissionPerItem << endl;
    cout << "Basic Salary: " << basicSalary << endl;
    cout << "Number of Subordinates: " << numOfSubordinates << endl;
    cout << "Salary: " << calculateSalary() << endl;
}

double calculateSalary() const {
    return (hoursWorked * ratePerHour) + (itemsSold * commissionPerItem) + basicSalary;
}
```

};

int main() {
WageEmployee wageEmp(1001, "John Doe", "1990-05-15", 40, 20.0);
wageEmp.display();
cout << endl;

```
SalesPerson salesPerson(2001, "Jane Smith", "1988-10-25", 100, 5.0);
salesPerson.display();
cout << endl;

Manager manager(3001, "Mike Johnson", "1975-03-08", 5000, 5);
manager.display();
cout << endl;

SalesManager salesManager(4001, "Sarah Davis", "1985-12-02", 200, 8.0, 8000, 10);
salesManager.display();
cout << endl;

return 0;
```

}

Q45

- Write a template class Stack (LIFO) having -

Date members – Size(int), Top of stack (int), Pointer of type T

Functionalities- Default constructor, push, pop, display

Test this code for pushing integers, doubles, complex objects.

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

template <class T>
class Stack {
private:
int size;
int top;
T* stackArray;

public:
Stack(int stackSize) {
size = stackSize;
top = -1;
stackArray = new T[size];
}
```

```
~Stack() {
    delete[] stackArray;
}

void push(const T& element) {
    if (top == size - 1) {
        cout << "Stack Overflow" << endl;
        return;
    }
    stackArray[++top] = element;
    cout << "Pushed: " << element << endl;
}

void pop() {
    if (top == -1) {
        cout << "Stack Underflow" << endl;
        return;
    }
    T poppedElement = stackArray[top--];
    cout << "Popped: " << poppedElement << endl;
}

void display() const {
    if (top == -1) {
        cout << "Stack is empty" << endl;
        return;
    }
    cout << "Stack elements: ";
    for (int i = top; i >= 0; i--) {
        cout << stackArray[i] << " ";
    }
    cout << endl;
}
```

};

int main() {
Stack<int> intStack(5);
intStack.push(10);
intStack.push(20);
intStack.push(30);
intStack.pop();
intStack.display();
cout << endl;

```
Stack<double> doubleStack(5);
doubleStack.push(3.14);
doubleStack.push(2.718);
doubleStack.push(1.618);
doubleStack.pop();
doubleStack.display();
cout << endl;

Stack<complex<double>> complexStack(5);
complex<double> c1(1.0, 2.0);
complex<double> c2(3.0, 4.0);
complex<double> c3(5.0, 6.0);
complexStack.push(c1);
complexStack.push(c2);
complexStack.push(c3);
complexStack.pop();
complexStack.display();
cout << endl;
```

```
return 0;
```

}

- Modify above class Stack in assignment 45 to handle “overflow” and “underflow” exceptions using exception handling feature.

#include <iostream>
using namespace std;

template <class T>
class Stack {
private:
int size;
int top;
T* stackArray;

public:
Stack(int stackSize) {
size = stackSize;
top = -1;
stackArray = new T[size];
}

```
~Stack() {
    delete[] stackArray;
}

void push(const T& element) {
    try {
        if (top == size - 1) {
            throw overflow_error("Stack Overflow");
        }
        stackArray[++top] = element;
        cout << "Pushed: " << element << endl;
    } catch (const overflow_error& e) {
        cerr << "Exception: " << e.what() << endl;
    }
}

void pop() {
    try {
        if (top == -1) {
            throw underflow_error("Stack Underflow");
        }
        T poppedElement = stackArray[top--];
        cout << "Popped: " << poppedElement << endl;
    } catch (const underflow_error& e) {
        cerr << "Exception: " << e.what() << endl;
    }
}

void display() const {
    if (top == -1) {
        cout << "Stack is empty" << endl;
        return;
    }
    cout << "Stack elements: ";
    for (int i = top; i >= 0; i--) {
        cout << stackArray[i] << " ";
    }
    cout << endl;
}
```

};

int main() {

Stack<int> intStack(3);

intStack.push(10);

intStack.push(20);

intStack.push(30);

intStack.push(40);

intStack.pop();

intStack.display();

cout << endl;

```
return 0;
```

}

Q47)

- Create a class Account with -

Instance variable - balance

Class functions - deposite/withdraw

User withdrawal limit on one transaction is Rs.15000/-

Throw and Handle Exceptions -

1. OverLimit - when user tries to withdraw more than Rs. 15000/- in a transaction
2. InsufficientBalance - When user withdrawal amount is more than existing balance

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

class OverLimitException : public exception {
public:
const char* message() const noexcept {
return "Withdrawal limit exceeded. Maximum limit is Rs. 15000/-";
}
};

class InsufficientBalanceException : public exception {
public:
const char* message() const noexcept {
return "Insufficient balance for withdrawal";
}
};

class Account {
private:
double balance;

public:
Account(double initialBalance = 0.0) {
balance = initialBalance;
}
```

```
void deposit(double amount) {
    balance += amount;
    cout << "Deposit: Rs. " << amount << endl;
}

void withdraw(double amount) {
    try {
        if (amount > 15000) {
            throw OverLimitException();
        }
        if (amount > balance) {
            throw InsufficientBalanceException();
        }
        balance -= amount;
        cout << "Withdrawal: Rs. " << amount << endl;
    } catch (const OverLimitException& e) {
        cerr << "Exception: " << e.message() << endl;
    } catch (const InsufficientBalanceException& e) {
        cerr << "Exception: " << e.message() << endl;
    }
}

double getBalance() const {
    return balance;
}
```

```
};
int main() {
Account account(10000.0);
account.deposit(5000.0);
account.withdraw(2000.0);
account.withdraw(25000.0); // Exceeds withdrawal limit
account.withdraw(15000.0); // Exceeds balance
cout << "Current Balance: Rs. " << account.getBalance() << endl;
```

```
return 0;
```

```
}
```

Q2) Write a class to represent a vector (a series of float values). Include member functions to perform the following
tasks:

(a) To create the vector

(b) To modify the value of a given element

(c) To multiply by a scalar value

(a) to display the vector in the form (10, 20, 30, ...)
Write a program to test your class.

```
#include <iostream>
#include <vector>

class Vector {
private:
std::vector<float> values;

public:
// Constructor to create the vector
Vector(const std::vector<float>& inputValues) : values(inputValues) {}
```

```
// Function to modify the value of a given element
void modifyElement(int index, float value) {
    if (index >= 0 && index < values.size()) {
        values[index] = value;
    } else {
        std::cout << "Invalid index" << std::endl;
    }
}

// Function to multiply the vector by a scalar value
void multiplyByScalar(float scalar) {
    for (float& value : values) {
        value *= scalar;
    }
}

// Function to display the vector
void displayVector() {
    std::cout << "(";
    for (const float& value : values) {
        std::cout << value << ", ";
    }
    std::cout << "\\b\\b)" << std::endl;
}
```

```
};

int main() {
Vector vector({10, 20, 30, 40});
vector.displayVector();  // Output: (10, 20, 30, 40)
vector.modifyElement(2, 35);
vector.displayVector();  // Output: (10, 20, 35, 40)

vector.multiplyByScalar(2);
vector.displayVector();  // Output: (20, 40, 70, 80)

return 0;
```

```
}
```