Structures in C++

1. Introduction to Structures

In C++, a **structure (struct)** is a user-defined data type that allows grouping multiple variables of different types under one name.

Structures are useful for defining complex data types that represent real-world entities, such as a Student, Car, or Book.

2. Declaring a Structure

A structure is declared using the struct keyword. It can contain variables (members) of different data types.

Syntax:

```
struct StructureName {
    data_type member1;
    data_type member2;
    ...
};
```

Example:

```
struct Student {
    string name;
    int age;
    float marks;
};
```

This structure defines a Student with three attributes: name, age, and marks.

3. Defining and Accessing Structure Members

Once a structure is declared, we can create variables (instances) of that structure and access its members using the **dot (.) operator**.

Example:

```
struct Student {
    string name;
    int age;
    float marks;
};
int main() {
    // Creating a structure variable
    Student s1;
     // Assigning values
    s1.name = "Alice";
    s1.age = 20;
    s1.marks = 85.5;
     // Accessing and displaying values
    cout << "Name: " << s1.name << endl;
    cout << "Age: " << s1.age << endl;
    cout << "Marks: " << s1.marks << endl;
    return 0;
}
```

4. Initializing Structures

We can initialize a structure in different ways.

Method 1: Using Direct Assignment

```
 Student s1 = {"Alice", 20, 85.5};
```

Method 2: Assigning Values After Declaration

```
 Student s2;
s2.name = "Bob";
s2.age = 22;
s2.marks = 90.2;
```

Method 3: Using Constructor (Inside Struct)

In C++, structures can have constructors.

```
 struct Student {
   string name;
   int age;
   float marks;
    // Constructor (Function)
   Student(string n, int a, float m) {
      name = n;
      age = a;
      marks = m;
   }
};
```

```
int main() {
   Student s3("Charlie", 21, 88.0);
   cout << "Name: " << s3.name << endl;
   cout << "Age: " << s3.age << endl;
   cout << "Marks: " << s3.marks << endl;
   return 0;
}
```

5. Array of Structures

We can create an array of structures to store multiple records.

Example:

```
struct Student {
    string name;
    int age;
    float marks;
};
```

```
int main() {
   Student students[2] = { {"Alice", 20, 85.5}, {"Bob", 22, 90.2} };

   for (int i = 0; i < 2; i++) {
      cout << "Student " << i+1 << ": " << students[i].name
          << ", Age: " << students[i].age
          << ", Marks: " << students[i].marks << endl;
   }
   return 0;
}
```

6. Pointer to Structure

We can use pointers to access structure members.

Example:

```
struct Student {
   string name;
   int age;
   float marks;
};
```

```
int main() {
   Student s1 = {"Alice", 20, 85.5};
   Student* ptr = &s1;

   // Accessing structure members using pointer
   cout << "Name: " << ptr->name << endl;
   cout << "Age: " << ptr->age << endl;
   cout << "Marks: " << ptr->marks << endl;

   return 0;
}
```

Here, ptr->name is the same as (*ptr).name.

7. Structure Object Inside Structure

A structure can contain another structure Object.

Example:

```
struct Address {
   string city;
   int zip;
};

struct Student {
   string name;
   int age;
   Address address;
};
```

```
int main() {
   Student s1 = {"Alice", 20, {"New York", 10001}};

   cout << "Name: " << s1.name << endl;
   cout << "City: " << s1.address.city << endl;
   cout << "ZIP: " << s1.address.zip << endl;

   return 0;
}
```

8. Structure with Functions

Functions can be used inside and outside structures.

Example 1: Function Inside Struct

```
struct Student {
    string name;
    int age;
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student s1 = {"Alice", 20};
    s1.display();
    return 0;
}
```

Example 2: Passing Structure to Function

```
struct Student {
    string name;
    int age;
};

void display(Student s) {
    cout << "Name: " << s.name << ", Age: " << s.age << endl;
}

int main() {
    Student s1 = {"Alice", 20};
    display(s1);
    return 0;
}
```

10. Conclusion

- **Structures** in C++ are used to group related data items.
- Unlike C, C++ structures can have **constructors, member functions, and access specifiers**.
- They are useful in organizing complex data.
- When more advanced features like **data hiding and inheritance** are needed, **classes** should be used.

1. Pointer Inside a Structure

A structure can contain a **pointer** as a member, which allows it to reference dynamically allocated memory or other structures.

Example: Pointer Inside Structure

```
struct Student {
   string name;
   int* age; // Pointer to an integer
};
```

```
int main() {
   Student s1;
   int a = 20;
   s1.name = "Alice";
   s1.age = &a; // Assigning address of a

   cout << "Name: " << s1.name << endl;
   cout << "Age: " << *(s1.age) << endl; // Dereferencing pointer

   return 0;
}
```

Basic structure using the struct keyword.

```
// Define a structure named "Person"
struct Person {
    string name;
    int age;
    double height;
    char gender;
};
```

```
int main() {
    // Create an instance of the "Person" structure
    Person person1;

    // Assign values to the structure members
    person1.name = "John Doe";
    person1.age = 30;
    person1.height = 6.0;
    person1.gender = 'M';

    // Display the information
    cout << "Name: " << person1.name << endl;
    cout << "Age: " << person1.age << endl;
    cout << "Height: " << person1.height << " feet" << endl;
    cout << "Gender: " << person1.gender << endl;

    return 0;
}
```

Basic structure using the struct keyword.

```
// Define a structure named "Point" for 2D coordinates

struct Point {
    double x;
    double y;
};
```

```
int main() {
    // Create an instance of the "Point" structure
    Point p1;

    // Assign values to the structure members
    p1.x = 3.5;
    p1.y = 2.0;

    // Display the coordinates
   cout << "Point coordinates: (" << p1.x << ", " << p1.y << ")" << endl;

    return 0;
}
```

Example of a C++ program with a structure containing an array as a member:

```cpp
// Define a struct named "Student" that contains an array of exam scores


struct Student {
    string name;
    int rollNumber;
    int examScores[3]; // Array to store exam scores for three exams
};
```

```cpp
int main() {
    // Create an instance of the "Student" struct
    Student student1;

    // Assign values to the structure members
    student1.name = "Alice";
    student1.rollNumber = 101;
    student1.examScores[0] = 85;
    student1.examScores[1] = 92;
    student1.examScores[2] = 78;

    // Display the student's information and exam scores
    cout << "Name: " << student1.name << endl;
    cout << "Roll Number: " << student1.rollNumber << endl;
    cout << "Exam Scores: ";
    for (int i = 0; i < 3; i++) {
        cout << student1.examScores[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Example of a C++ structure with an array of structures:

```
// Define a structure named
"Student" to represent
student information

struct Student {
   string name;
   int rollNumber;
};
```

```
int main() {
 // Create an array of "Student" structures
   const int numStudents = 3; // Number of students
   Student students[numStudents];

   // Assign values to the structure members for each student
   students[0].name = "Alice";
   students[0].rollNumber = 101;

   students[1].name = "Bob";
   students[1].rollNumber = 102;

   students[2].name = "Charlie";
   students[2].rollNumber = 103;

   // Display the information for each student in the array
   for (int i = 0; i < numStudents; i++) {
      cout << "Student " << i + 1 << " Information:" << endl;
      cout << "Name: " << students[i].name << endl;
      cout << "Roll Number: " << students[i].rollNumber << endl;
      cout << endl;
   }
   return 0;
}
```

Example of a C++ program with a nested struct.

```
// Define a struct named "Address"
for storing address information

struct Address {
    string street;
    string city;
    string state;
    string zipCode;
};

// Define a struct named "Person"
that includes the "Address" struct as
a member

struct Person {
    string name;
    int age;
    Address address;
};
```

```
int main() {
    // Create an instance of the "Person" struct
    Person person1;

    // Assign values to the structure members
    person1.name = "John Doe";
    person1.age = 30;
    person1.address.street = "123 Main St";
    person1.address.city = "Anytown";
    person1.address.state = "CA";
    person1.address.zipCode = "12345";

    // Display the person's information, including the nested "Address" struct
    cout << "Name: " << person1.name << endl;
    cout << "Age: " << person1.age << endl;
    cout << "Address:" << endl;
    cout << "Street: " << person1.address.street << endl;
    cout << "City: " << person1.address.city << endl;
    cout << "State: " << person1.address.state << endl;
    cout << "Zip Code: " << person1.address.zipCode << endl;

    return 0;
}
```

Passing Structure Members as Arguments to Functions:

You can pass individual members of a structure as arguments to a function. For example:

```
struct Point {
    int x;
    int y;
};

void printCoordinates(int x, int y) {
    cout << "X: " << x << ", Y: " << y << endl;
}

Point myPoint = {5, 10};

printCoordinates(myPoint.x, myPoint.y);
```

Passing Structure Variables as Parameters:

You can pass entire structure variables as function parameters:

```
void printPoint(Point p) {
    cout << "X: " << p.x << ", Y: " << p.y << endl;
}

Point myPoint = {5, 10};

printPoint(myPoint);
```

Returning Structure from Function:

Functions can return structures as well:

```
Point createPoint(int x, int y) {
    Point p;
    p.x = x;
    p.y = y;
    return p;
}

Point newPoint = createPoint(3, 7);
```

Pointers to Structure Variables:

You can use pointers to access and manipulate structure variables:

```
Point myPoint = {5, 10};

Point* pPoint = &myPoint;

cout << "X: " << pPoint->x << ", Y: " << pPoint->y << endl;
```

Passing Structure Pointers as Arguments to a Function:

You can pass pointers to structures as function parameters:

```
void modifyPoint(Point* p) {
    p->x += 2;
    p->y += 2;
}

Point myPoint = {5, 10};

modifyPoint(&myPoint);
```

Returning a Structure Pointer from Function:

Functions can also return pointers to structures:

```
Point* createAndReturnPoint(int x, int y) {
    Point* p = new Point;
    p->x = x;
    p->y = y;
    return p;
}

Point* newPoint = createAndReturnPoint(3, 7);
```

```
struct Point {
    int x, y;
};

Point* createAndReturnPoint(int x, int y) {
   Point* p = new Point;
   p->x = x;
   p->y = y;
   return p;
}

int main() {
   Point* newPoint = createAndReturnPoint(3, 7);

   cout << "Point: (" << newPoint->x << ", " << newPoint->y << ")\n";

   // Free allocated memory
   delete newPoint;

   return 0;
}
```

Passing Array of Structures:

You can create an array of structures and pass them to functions:

```
struct Student {
    string name;
    int age;
};

void printStudents(Student students[], int size) {
    for (int i = 0; i < size; i++) {
        cout << "Name: " << students[i].name << ", Age: " << students[i].age << endl;
    }
}

Student classStudents[3] = {{"Alice", 20}, {"Bob", 22}, {"Charlie", 19}};

printStudents(classStudents, 3);
```

Dynamic allocation within a struct typically involves allocating memory for one or more members of the struct using pointers. This is commonly used when you need to handle variable-sized data or when you want to manage memory manually.

```
// Define a struct named
"Student" that includes
dynamic memory allocation


struct Student {
// Dynamic memory allocation for
name
char* name;

int rollNumber;
};
```

```
int main() {
   // Create an instance of the "Student" struct
   Student student1;

   // Allocate memory for the name member dynamically
   student1.name = new char[50]; // Allocates space for a name of up to 49 characters

   // Assign values to the structure members
   cout << "Enter Name: ";
   cin.getline(student1.name, 50);

   cout << "Enter Roll Number: ";
   cin >> student1.rollNumber;

   // Display the student's information
   cout << "Name: " << student1.name << endl;
   cout << "Roll Number: " << student1.rollNumber << endl;

   // Don't forget to release the allocated memory when you're done
   delete[] student1.name;

   return 0;
}
```

Dynamic allocation of a 2D array within a struct involves using pointers to create a dynamically allocated 2D array and then storing a pointer to this array as a member of the struct.

```
// Define a struct named "Matrix" to store a dynamic 2D array


struct Matrix {
 // Pointer to a dynamically allocated 2D array

   int** data;

   int rows;
   int cols;
};
```

```
// Function to allocate memory for a dynamic 2D array
int** createDynamic2DArray(int rows, int cols) {
   int** array = new int*[rows]; // Allocate memory for an array of int pointers (rows)
   for (int i = 0; i < rows; i++) {
      array[i] = new int[cols]; // Allocate memory for each row (cols)
   }
   return array;
}


// Function to deallocate memory for a dynamic 2D array
void deleteDynamic2DArray(int** array, int rows) {
   for (int i = 0; i < rows; i++) {
      delete[] array[i]; // Deallocate memory for each row
   }
   delete[] array; // Deallocate memory for the array of int pointers
}
```

```
int main() {
// Create an instance of the "Matrix" struct
    Matrix matrix1;

    // Input the number of rows and columns
    cout << "Enter the number of rows: ";
    cin >> matrix1.rows;
    cout << "Enter the number of columns: ";
    cin >> matrix1.cols;

// Allocate memory for the dynamic 2D array
matrix1.data = createDynamic2DArray(matrix1.rows, matrix1.cols);

// Input data into the matrix
    cout << "Enter matrix elements:" << endl;
    for (int i = 0; i < matrix1.rows; i++) {
        for (int j = 0; j < matrix1.cols; j++) {
            cin >> matrix1.data[i][j];
        }
    }
```

```
    // Display the matrix
    cout << "Matrix:" << endl;
    for (int i = 0; i < matrix1.rows; i++) {
        for (int j = 0; j < matrix1.cols; j++) {
            cout << matrix1.data[i][j] << " ";
        }
        cout << endl;
    }


// Deallocate memory when you're done
deleteDynamic2DArray(matrix1.data, matrix1.rows);

    return 0;
}
```

Dynamic Memory Allocation (DMA) is a technique in C++ that allows you to allocate memory for variables at runtime from the heap memory. When dealing with structures, you can dynamically allocate memory for structure variables using pointers.

```
// Define a
structure

struct Student
{
   string name;
   int age;
};
```

```
int main() {
   // Dynamically allocate memory for a single structure variable
   Student* studentPtr = new Student;

   // Initialize the dynamically allocated structure
   studentPtr->name = "Alice";
   studentPtr->age = 20;

// Access and print the data
cout << "Name: " << studentPtr->name << ", Age: " << studentPtr->age << endl;

   // Don't forget to deallocate the memory when done
   delete studentPtr;
// Dynamically allocate memory for an array of structure variables
    int numStudents = 3;
    Student* studentArray = new Student[numStudents];

    // Initialize the dynamically allocated array
    studentArray[0] = {"Bob", 22};
    studentArray[1] = {"Charlie", 19};
    studentArray[2] = {"David", 21};

    // Access and print the data in the array
    for (int i = 0; i < numStudents; i++) {
        cout << "Name: " << studentArray[i].name << ", Age: " << studentArray[i].age << endl;
    }

    // Don't forget to deallocate the memory when done
    delete[] studentArray;

    return 0;
}
```

Functions within structures

```
struct Person {
   string name;
   int age;
   double height;

// Member function to initialize a Person object

 void initialize(const string& n, int a, double h) {
       name = n;
       age = a;
       height = h;
   }

// Member function to display information about the person

void display() {
cout << "Name: " << name << endl;
cout << "Age: " << age << endl;
cout << "Height: " << height << endl;
   }
};
```

```
int main()
{
   // Create a Person object

Person person1;

 // Call the initialize function to set the values

person1.initialize("John", 30, 6.1);

// Call the display function to show information

person1.display();

   return 0;
}
```

Functions within structures

```
struct Rectangle {

    double length;
    double width;

// Member function to calculate the area of the rectangle
    double calculateArea() {
        return length * width;
    }

// Member function to calculate the perimeter of the rectangle
    double calculatePerimeter() {
        return 2 * (length + width);
    }

 // Member function to display information about the rectangle
void displayInfo()  {
cout << "Length: " << length << endl;
cout << "Width: " << width << endl;
cout << "Area: " << calculateArea() << endl;
cout<<"Perimeter:" <<calculatePerimeter()<<endl;
    }
};
```

```
int main() {

    // Create a Rectangle object

Rectangle myRectangle;

    // Set the dimensions of the
rectangle

myRectangle.length = 5.0;
myRectangle.width = 3.0;

// Display information about the
rectangle

myRectangle.displayInfo();

    return 0;
}
```

Functions within structures

```
struct Rectangle {
   double length;
   double width;
// Function prototypes inside the structure
   double calculateArea();
   double calculatePerimeter();
   void displayInfo();
};

// Function to calculate the area of a rectangle given a Rectangle object
double Rectangle::calculateArea() {
   return length * width;
}
// Function to calculate the perimeter of a rectangle given a Rectangle object
double Rectangle::calculatePerimeter() {
   return 2 * (length + width);
}
// Function to display information about a rectangle given a Rectangle object
void Rectangle::displayInfo() {
   cout << "Length: " << length << endl;
   cout << "Width: " << width << endl;
   cout << "Area: " << calculateArea() << endl;
   cout << "Perimeter: " << calculatePerimeter() << endl;
}
```

```
int main() {

   // Create a Rectangle object

   Rectangle myRectangle;

   // Set the dimensions of the rectangle

   myRectangle.length = 5.0;
   myRectangle.width = 3.0;

   // Display information about the rectangle using the functions

   myRectangle.displayInfo();

   return 0;
}
```

1. Create a student structure, whose members are
    i. Name (a char array),
    ii. roll_number,
    iii. marks (an array of type float having size 5),
    iv. major (a char array, to show the major of the student).
2. There shall also be a nested structure of type date struct inside the student structure, for the birthdate and registration date.
3. Now first create a student variable named CSStudent.
4. Fill up all the fields (members) with some random values from the console using “cin”
5. Secondly create another student variable named EEStudent.
6. Assign CSStudent to EEStudent.
7. Show the values of the members of both struct variables using cout.

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

// Define a structure for representing dates
struct Date {
    int day;
    int month;
    int year;
};

// Define a structure named "Student"
struct Student {
    char name[50];
    int roll_number;
    float marks[5];
    char major[50];
    Date birthdate;
    Date registration_date;
};
```

```
int main() {
    // Create a variable named "CSStudent" of type "Student"
    Student CSStudent;

    // Input values for CSStudent from the console using "cin"
    cout << "Enter Name: ";
    cin.getline(CSStudent.name, sizeof(CSStudent.name));

    cout << "Enter Roll Number: ";
    cin >> CSStudent.roll_number;

    cout << "Enter Marks for 5 Subjects: ";
    for (int i = 0; i < 5; i++) {
        cin >> CSStudent.marks[i];
    }

    cin.ignore(); // Ignore the newline character left in the input buffer
```

```
 cout << "Enter Major: ";
 cin.getline(CSStudent.major, sizeof(CSStudent.major));

 cout << "Enter Birthdate (day month year): ";
 cin >> CSStudent.birthdate.day;
 cin >> CSStudent.birthdate.month;
 cin >> CSStudent.birthdate.year;

 cout << "Enter Registration Date (day month year): ";
 cin >> CSStudent.registration_date.day;
cin >> CSStudent.registration_date.month;
cin >> CSStudent.registration_date.year;
```

```
// Create another student variable named "EEStudent" and assign CSStudent to it
    Student EEStudent = CSStudent;

    // Display the values of CSStudent and EEStudent
    cout << "\nValues of CSStudent:" << endl;
    cout << "Name: " << CSStudent.name << endl;
    cout << "Roll Number: " << CSStudent.roll_number << endl;
    cout << "Marks: ";
    for (int i = 0; i < 5; i++) {
        cout << CSStudent.marks[i] << " ";
    }
    cout << endl;
    cout << "Major: " << CSStudent.major << endl;
    cout << "Birthdate: " << CSStudent.birthdate.day << "/" <<
CSStudent.birthdate.month << "/" << CSStudent.birthdate.year << endl;
    cout << "Registration Date: " << CSStudent.registration_date.day << "/" <<
CSStudent.registration_date.month << "/" << CSStudent.registration_date.year <<
endl;
```

```
    cout << "\nValues of EEStudent (assigned from CSStudent):" << endl;
    cout << "Name: " << EEStudent.name << endl;
    cout << "Roll Number: " << EEStudent.roll_number << endl;
    cout << "Marks: ";
    for (int i = 0; i < 5; i++) {
        cout << EEStudent.marks[i] << " ";
    }
    cout << endl;
    cout << "Major: " << EEStudent.major << endl;
    cout << "Birthdate: " << EEStudent.birthdate.day << "/" <<
EEStudent.birthdate.month << "/" << EEStudent.birthdate.year << endl;
    cout << "Registration Date: " << EEStudent.registration_date.day << "/" <<
EEStudent.registration_date.month << "/" << EEStudent.registration_date.year <<
endl;

    return 0;
}
```

- **Define and Print a Structure:**
  Write a C++ program to define a struct Student with members name, age, and grade. Create an instance and print its values.

- **User Input in Structure:**
  Modify the previous program to take input from the user and display the details of a student.

- **Array of Structures:**
  Create an array of struct Book containing title, author, and price. Store details of 3 books and display them.

- **Function with Structure Argument:**
  Write a function that takes a struct Rectangle with length and width as arguments and returns the area.

- **Structure with Default Values:**
  Define a structure Car with brand, model, and year. Initialize it using default values inside main().

- **Pass Structure by Reference:**
Create a struct Employee with name, salary, and designation. Write a function that modifies the salary by reference.

- **Nested Structures:**
Define a structure Address inside struct Employee. Store city and state within Address. Create an employee instance and print its details.

- **Pointer to Structure:**
Create a pointer to a struct Student, dynamically allocate memory, assign values, and display them.

- **Structure with Array Member:**
Define struct Exam containing subject[3] and marks[3]. Store three subjects and marks for a student, then print them.

- **Dynamic Array of Structures:**
Write a program to dynamically allocate an array of struct Employee, take user input for multiple employees, and print their details.

- **Structure and Sorting:**
Define struct Student with name and marks. Store details of five students in an array and sort them in descending order of marks.