Pointers and Structure in C Plus Plus

Module 2 - Structure, Union and Enumeration in C Plus Plus

Structures in C++

A **structure** in C++ is a user-defined data type that groups together variables (which can be of different types) under a single name. Structures are used to represent a record. For example, a student's record might include their name, age, and marks.

Structures are useful when you want to store related data together, especially when that data is of different types.

Syntax for Structure Declaration

```
struct StructureName {
    // Data members (variables)
    data_type member1;
    data_type member2;
    //...
};
```

- struct: Keyword to define a structure.
- StructureName: The name of the structure (can be anything).
- member1, member2, ...: Data members of the structure.

Solved Example 1: Basic Structure Declaration

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

// Structure definition
struct Student {
    string name;
    int age;
    float marks;
};

int main() {
    // Creating a structure variable
    Student student1;

    // Assigning values to the structure members
    student1.name = "John";
    student1.age = 20;
    student1.marks = 85.5;

    // Accessing and printing the structure members
    cout << "Name: " << student1.name << endl;
    cout << "Age: " << student1.age << endl;
    cout << "Marks: " << student1.marks << endl;

    return 0;
}
```

Explanation:

- struct Student: Defines a structure named Student with three members: name, age, and marks.
- We create a variable student1 of type Student.
- We then assign values to each member of the structure and print them.

Output:

Name: John
Age: 20
Marks: 85.5

Solved Example 2: Structure with Functions

Structures can also be used with functions. You can pass a structure to a function, return a structure from a function, or access its members within the function.

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

// Structure definition
struct Rectangle {
    float length;
    float width;
};

// Function to calculate the area of a rectangle
float calculateArea(Rectangle rect) {
    return rect.length * rect.width;
}

int main() {
    // Creating and initializing a structure variable
    Rectangle rect1 = {5.0, 3.0};

    // Calling the function with the structure as argument
    float area = calculateArea(rect1);

    // Printing the area
    cout << "Area of the rectangle: " << area << endl;

    return 0;
}
```

Explanation:

- We define a Rectangle structure with two members: length and width.
- The calculateArea function takes a Rectangle as an argument and computes its area.
- The result is printed in main.

Output:

Area of the rectangle: 15

Assignment

Ques 1: Basic Structure
Create a structure called Book with the following members:
- title (string)
- author (string)
- price (float)

Then, write a program to:
- Declare a variable of type Book
- Assign values to each member
- Print the details of the book.

Ques 2: Structure with Functions
Define a structure Circle with two members:
- radius (float)
- color (string)

Write a function area() that calculates the area of the circle. Pass the structure to the function, calculate the area and print it in the main() function.

Union in C++

In C++, a **union** is a special data type that allows storing different data types in the same memory location. Unlike a structure, where each member has its own memory location, a union uses a single shared memory for all of its members. This means that at any given time, a union can hold only one of its members' values.

Syntax of Union

```
union UnionName {
    dataType member1;
    dataType member2;
    dataType member3;
};
```

Size of a Union

The size of a union is the size of its largest member. In the example above, if int takes 4 bytes, float takes 4 bytes, and char takes 1 byte, the size of the union Data will be 4 bytes because the largest member is an int or float (both 4 bytes).

Difference between Structure and Union in C++

Key Differences:

1. **Memory Allocation:**
   - **Structure**: In a structure, each member has its own memory. So, if you have 3 members (e.g., an int, float, and char), they each get their own space.
   - **Union**: In a union, all members share the same memory space. Only one member can hold a value at a time. The memory is large enough to store the largest member.
2. **Access to Members:**
   - **Structure**: You can access and store values for all members at the same time. Each member's value is independent of others.
   - **Union**: You can only access one member at a time because all members share the same memory space. Changing one member will overwrite the previous one.
3. **Memory Size:**
   - **Structure**: The size of a structure is the sum of the sizes of all its members.
   - **Union**: The size of a union is the size of its largest member (since they share the same memory).

Enumeration in C++

In C++, an **enum** (short for **enumeration**) is a user-defined data type that allows you to assign names to integral values. It makes your code more readable by giving meaningful names to numbers, rather than using raw numeric values.

Key Points:

1. **Purpose**: An enum is used to define a set of constants, typically representing a collection of related values.
2. **Default Values**: By default, the first name in an enum is assigned the value 0, and each subsequent name is assigned an incremented integer value (1, 2, 3, ...). You can also manually assign specific values to the names.
3. **Type**: Enums in C++ are typically represented as int values, but they can be explicitly defined to use a different integer type.

Syntax:

```
enum EnumName {
    Constant1,
    Constant2,
    Constant3
};
```

Introduction to 'C++' Language

Solved Example of Enum in C++

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

enum Days {
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
};

int main() {
    Days today = Wednesday;  // Assigning a value from the enum to a variable

    if (today == Wednesday) {
        cout << "Today is Wednesday." << endl;
    }

    // You can also print the numeric value of an enum member
    cout << "Numeric value of Sunday: " << Sunday << endl;  // Output: 0
    cout << "Numeric value of Wednesday: " << Wednesday << endl;  // Output: 3

    return 0;
}
```

Explanation:

1. **Enum Declaration**: The enum Days represents the days of the week. By default, Sunday is assigned 0, Monday 1, and so on.
2. **Assigning Enum Values**: In the main() function, we assign the value Wednesday to the variable today.
3. **Comparing Enum Values**: We check if today is equal to Wednesday and print a message accordingly.
4. **Printing Enum as Integer**: You can print the numeric value associated with an enum by simply using the enum variable in the output statement.

Output:

```
Today is Wednesday.
Numeric value of Sunday: 0
Numeric value of Wednesday: 3
```