Parul® University

303105151 - Computational Thinking for Structured Design-2

Miss. Miral Maradiya

Assistant Professor

Computer Science & Engineering

CHAPTER-3

Enumerators, Structures, Unions

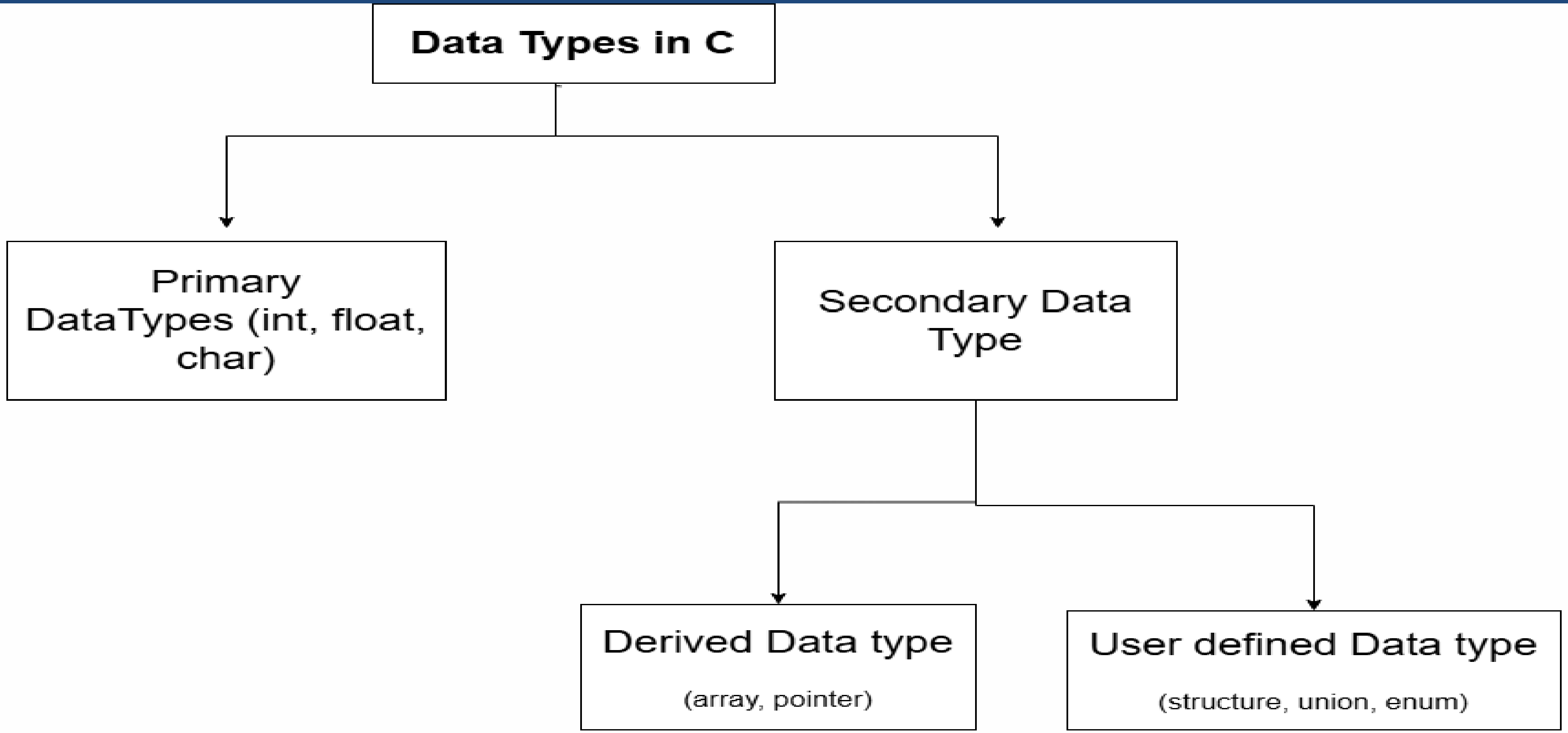
Contents

1. Enumerators
2. Structures
3. Union
4. Typedef

Data Types

- Data types are defined as the data storage format that a variable can store a data.
- C language has built-in datatypes like primary and derived data types.
- But, still not all real world problems can be solved using those data types.
- We need custom datatype for different situation.

Data Types



Data Types in C

Primary DataTypes (int, float, char)

Secondary Data Type

Derived Data type

(array, pointer)

User defined Data type

(structure, union, enum)

User Defined Datatype

We need combination of various datatypes to understand different entity/object.

Example-1:

Book

Title: Let Us C	//Datatype: char / string
Author: Yashavant Kanetkar	//Datatype: char / string
Page: 320	//Datatype: int
Price: 255.00	//Datatype: float

Example-2:

Student

Name: ABC	//Datatype: char / string
Roll_No: 180540107001	//Datatype: int
CPI: 7.46	//Datatype: float
Backlog: 01	//Datatype: int

1. What is an Enumerator in C?

- Definition:

  An enumerator in C is a user-defined data type consisting of integral constants.

- Purpose:

  Provides a way to assign names to a set of numeric values for better code readability and maintainability.

- Declaration Syntax:

```
enum enum_name {constant1, constant2, ...};
```

Features of Enumerators in C

- **Integral Constants:**

  Enumerators are represented by integers, starting from 0 by default.
- **Named Constants:**

  Improves clarity by avoiding the use of hard-coded numbers.
- **Scoped Values:**

  Enumerators are scoped within the enum definition, preventing naming conflicts.
- **Custom Values:**

  You can assign specific integer values to individual constants.

  Example : enum colors {RED = 1, GREEN, BLUE};

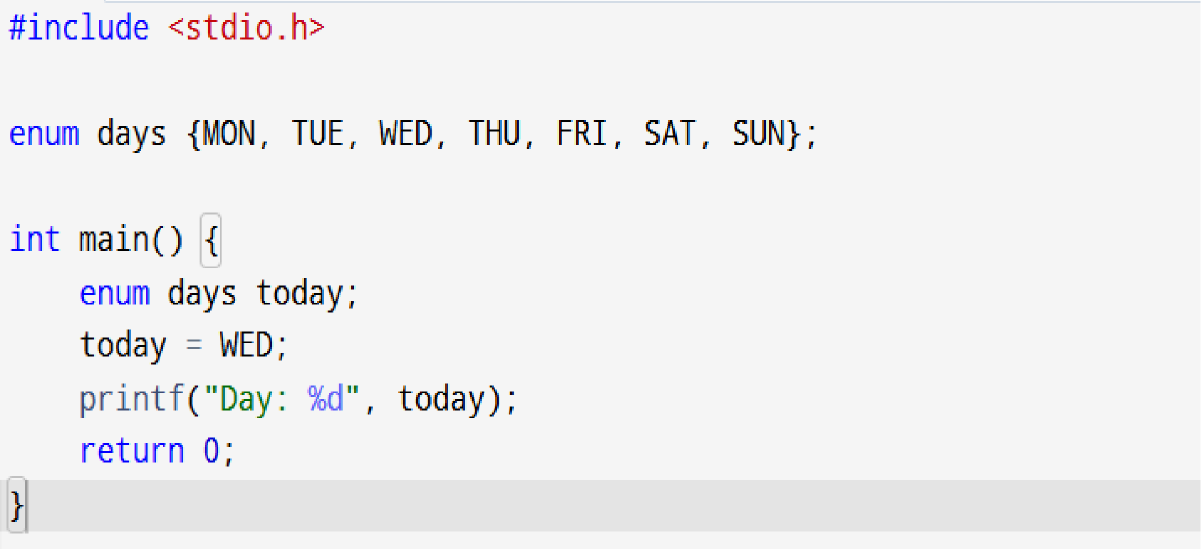
  In this case, GREEN will be 2, and BLUE will be 3.

Declaring and Using Enumerators

Declaration Example:

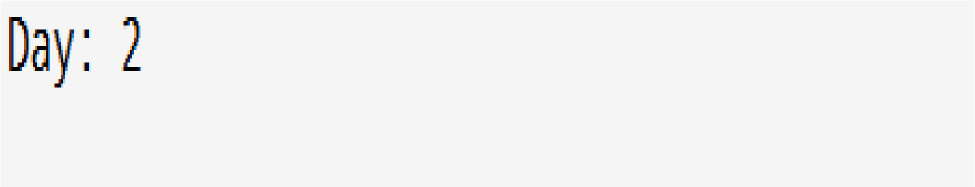
enum days {MON, TUE, WED, THU, FRI, SAT, SUN};

Usage in Code:



```c
#include <stdio.h>

enum days {MON, TUE, WED, THU, FRI, SAT, SUN};

int main() {
    enum days today;
    today = WED;
    printf("Day: %d", today);
    return 0;
}
```

Output :



```
Day: 2
```

Advantages of Enumerators in C

- **Improved Readability:**
  Meaningful names make the code self-explanatory.
- **Type Safety:**
  Ensures variables of enum types only hold predefined values.
- **Ease of Maintenance:**
  Makes updates easier when adding or modifying constants.
- **Memory Efficiency:**
  Enum values use the same memory as integers, making them lightweight.

Limitations of Enumerators

- **Limited to Integers:**
  Enum values must be integral constants, limiting flexibility.

- **No String Representation:**
  Requires additional code to map enum values to descriptive strings.

- **Compiler-Specific Behavior:**
  Enum size and implementation may vary between compilers.

Practical Applications

- **State Management:**

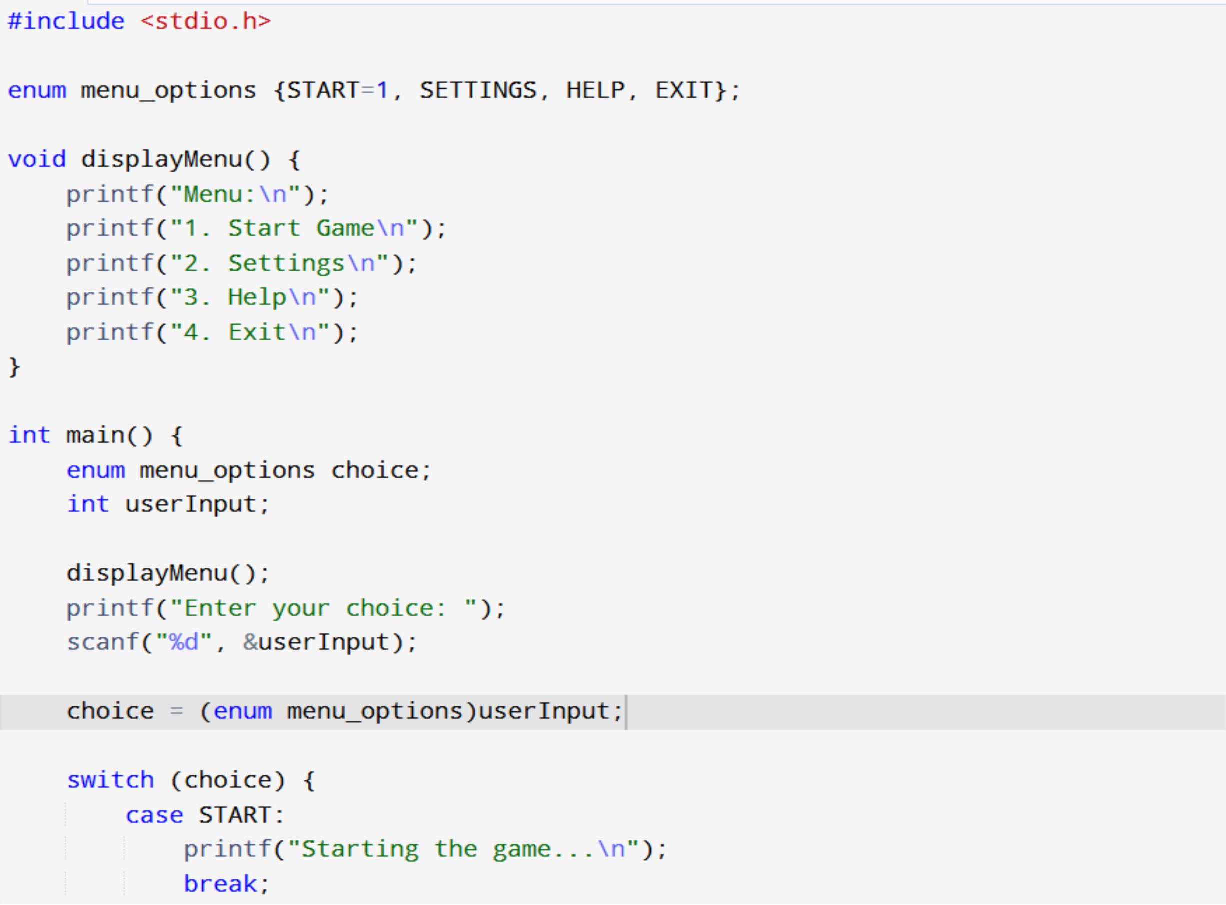
  Define states like START, PROCESSING, COMPLETED.
- **Error Codes:**

  Use enums to categorize error messages systematically.
- **Menu Options:**

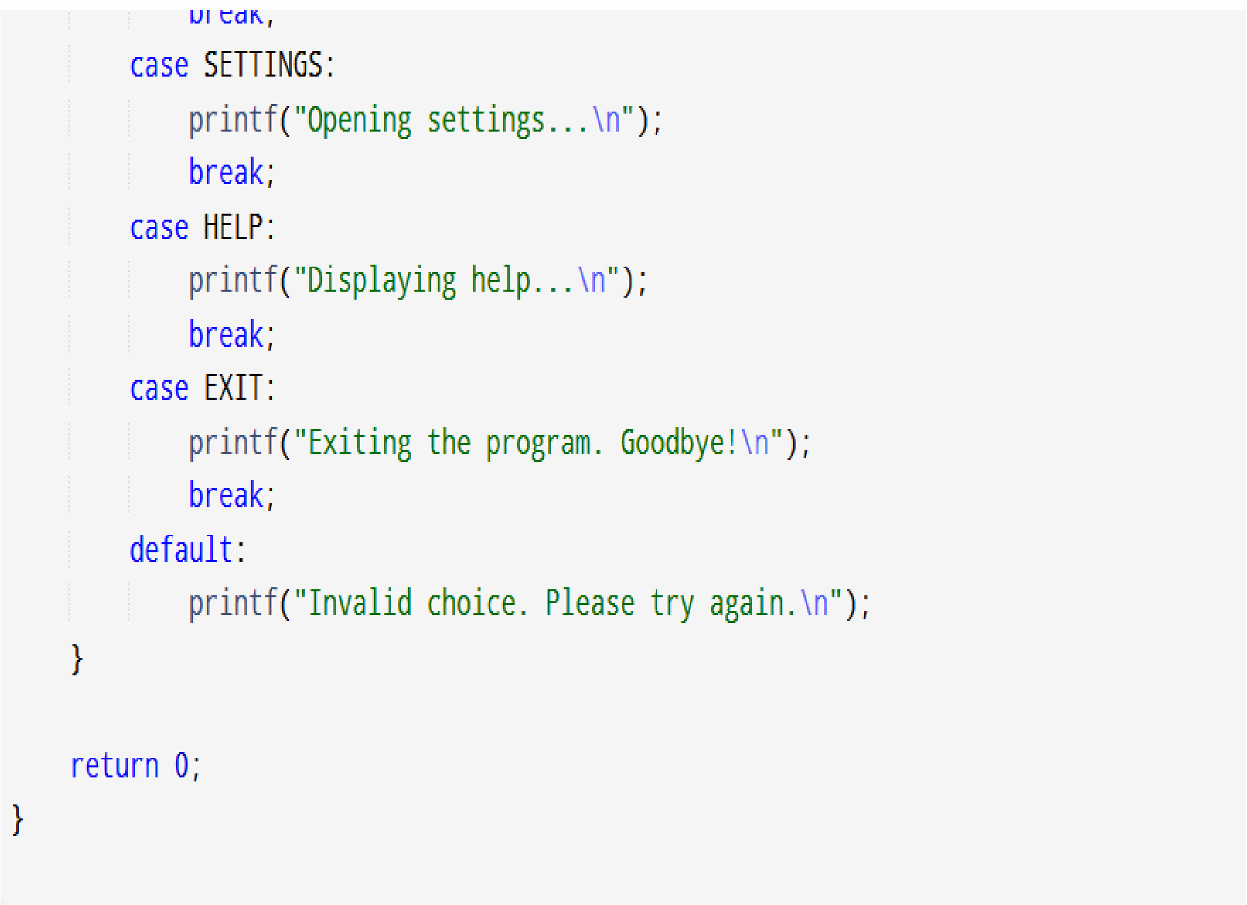
  Enumerate options for user interfaces or configuration menus.
- **Flags:**

  Represent feature toggles or settings.

Example :



```c
#include <stdio.h>

enum menu_options {START=1, SETTINGS, HELP, EXIT};

void displayMenu() {
    printf("Menu:\n");
    printf("1. Start Game\n");
    printf("2. Settings\n");
    printf("3. Help\n");
    printf("4. Exit\n");
}

int main() {
    enum menu_options choice;
    int userInput;

    displayMenu();
    printf("Enter your choice: ");
    scanf("%d", &userInput);

    choice = (enum menu_options)userInput;

    switch (choice) {
        case START:
            printf("Starting the game...\n");
            break;
```



```c
            break;
        case SETTINGS:
            printf("Opening settings...\n");
            break;
        case HELP:
            printf("Displaying help...\n");
            break;
        case EXIT:
            printf("Exiting the program. Goodbye!\n");
            break;
        default:
            printf("Invalid choice. Please try again.\n");
    }

    return 0;
}
```

Enum Exercises :

- Define an enum Season with values WINTER, SPRING, SUMMER, and AUTUMN. Write a program that asks the user to enter a month number (1 for January, 2 for February, etc.) and prints the corresponding season.
- Define an enum for different access rights: READ = 1, WRITE = 2, EXECUTE = 4, DELETE = 8. Allow the user to select one or more rights by entering a combination of values. Then, use bitwise operations to print the corresponding access rights.
- Define an enum for a simple menu system: START, STOP, PAUSE, RESUME. Prompt the user for a menu choice and use a switch statement to display a corresponding message. Ensure that invalid inputs are handled gracefully by printing an error message.

2. Structure

- A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. (Structures are called “records” in some languages, notably Pascal.)
- Structure is a user defined datatype.
- Structures help to organize complicated data, particularly in large programs, because they permit a group of related variables to be treated as a unit instead of as separate entities.

Structure Vs. Array

Aspect	Array	Structure
Data Type	Same type (homogeneous)	Different types (heterogeneous)
Memory Allocation	Contiguous (fixed size)	Sequential (with possible padding)
Access	Using index ([])	Using member name (.)
Size	Fixed at compile time	Sum of member sizes, with possible padding
Use Case	Lists of similar data	Grouping of different types of data

Syntax of Structure Declaration

- Two fundamental aspects of Structure:
  - Declaration of Structure Variable
  - Accessing of Structure Member

Syntax of structure :

```
struct StructName {
   dataType member1;
   dataType member2;
   // More members...
};
```

Syntax of Structure Declaration

- To define a structure, we need to use struct keyword.
- This keyword is reserved word in C language.
- We can only use it for structure and its object declaration.
- Members can be normal variables, pointers, arrays or other structures.
- Member names within the particular structure must be distinct from one another.
- You must terminate structure definition with semicolon ;.

Example of Structure Declaration

Example : 1

```
struct student
{
char name[30]; // Student Name
int roll_no; // Student Roll No
float CPI; // Student CPI
int backlog; // Student Backlog
};
```

Example : 2

```
struct Person {
   char name[50];  // A string to store the name
   int age;        // An integer to store the age
   float height;   // A float to store the height
};
```

- You cannot assign value to members inside the structure definition, it will cause compilation error.

Example :

```
struct student
{
   char name[30] = “ABC”; // Student Name
   . . .
};
```

Create Structure variable

- A data type defines various properties about data stored in memory.
- To use any type we must declare its variable.
- Hence, let us learn how to create our custom structure type objects also known as structure variable.
- In C programming, there are two ways to declare a structure variable:
  1. Along with structure definition
  2. After structure definition

Create Structure variable

Method	Along with Structure Definition	After Structure Definition

Syntax	struct StructureName { ... } variable1, variable2;	struct StructureName { ... };
		struct StructureName variable1, variable2;

Where Structure Variables are Declared	The structure variables are declared immediately after the structure definition.	The structure variables are declared separately after the structure is defined.
When to Use	Useful when you want to define and initialize structure variables at the same time.	Useful when you want to define the structure first and declare the variables later.

Create Structure variable

1. Declaring Structure Variables Along with Structure Definition:

Syntax :

```
struct StructureName {
   dataType member1;
   dataType member2;
   // More members...
} variable1, variable2, ...;
```

Example :

```
struct Person {
   char name[50];
   int age;
   float height;
      } person1, person2;  // Declare person1 and
                           person2 as structure variables
```

Create Structure variable

2. Declaration after Structure definition

Syntax :

```
struct StructureName {
   dataType member1;
   dataType member2;
   // More members...
};

struct StructureName variable1, variable2, ...;
// Declare variables later
```

Example :

```
struct Person {
   char name[50];
   int age;
   float height;
};

struct Person person1, person2;
// Declare structure variables after the definition
```

Access Structure member (data)

- Structure is a complex data type, we cannot assign any value directly to it using assignment operator.
- We must assign data to individual structure members separately.
- C supports two operators to access structure members, using a structure variable:

1. Dot/period operator (.)
2. Arrow operator (->)

Access Structure member (data)

1. Dot/period operator (.)

It is known as member access operator. We use dot operator to access members of simple structure variable.

Syntax :

```
structure_variable.member_name;
```

Example :

```
// Assign CPI of student1
student1.CPI = 7.46;
```

Access Structure member (data)

2. Arrow operator (->)

In C language it is illegal to access a structure member from a pointer to structure variable using dot operator.

- We use arrow operator to access structure member from pointer to structure.

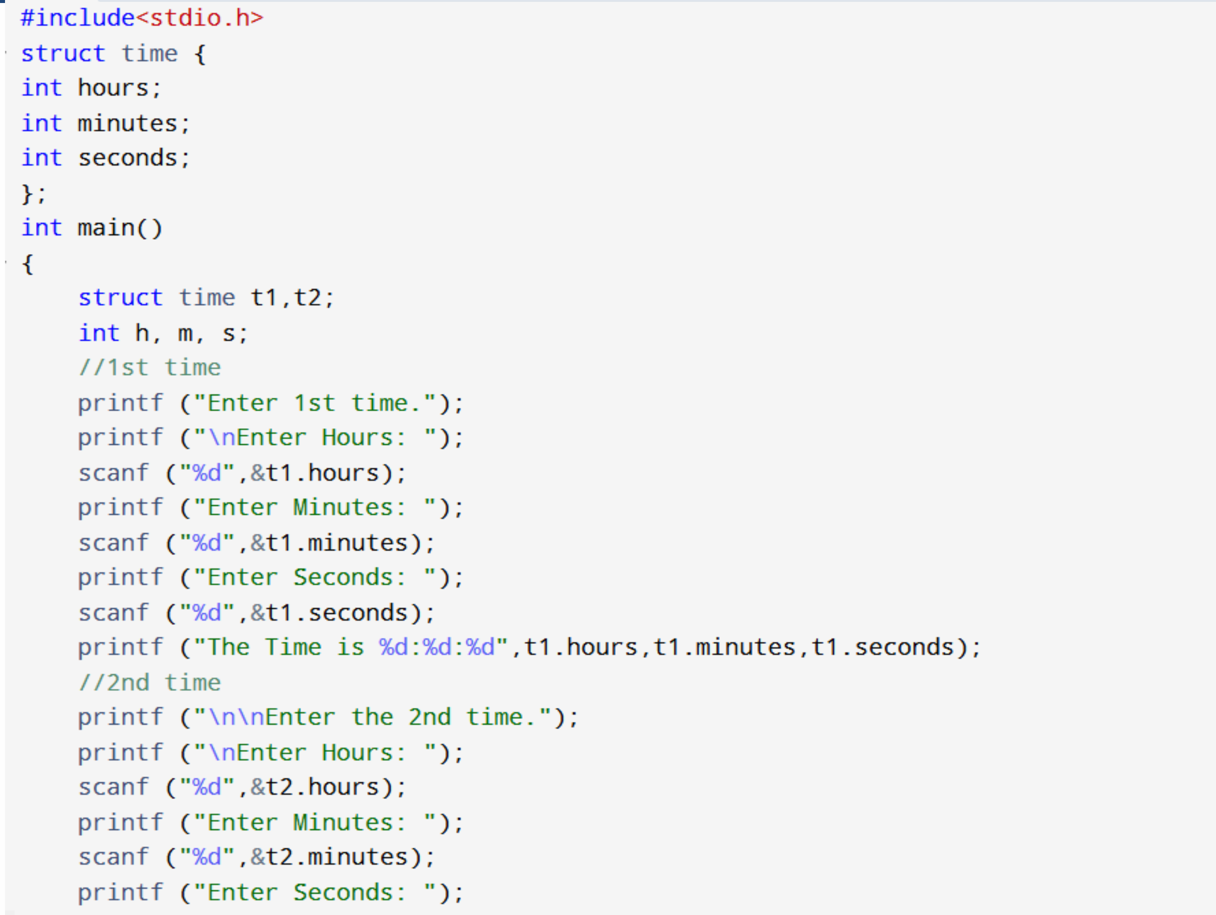
Syntax :

```
pointer_to_structure->member_name;
```

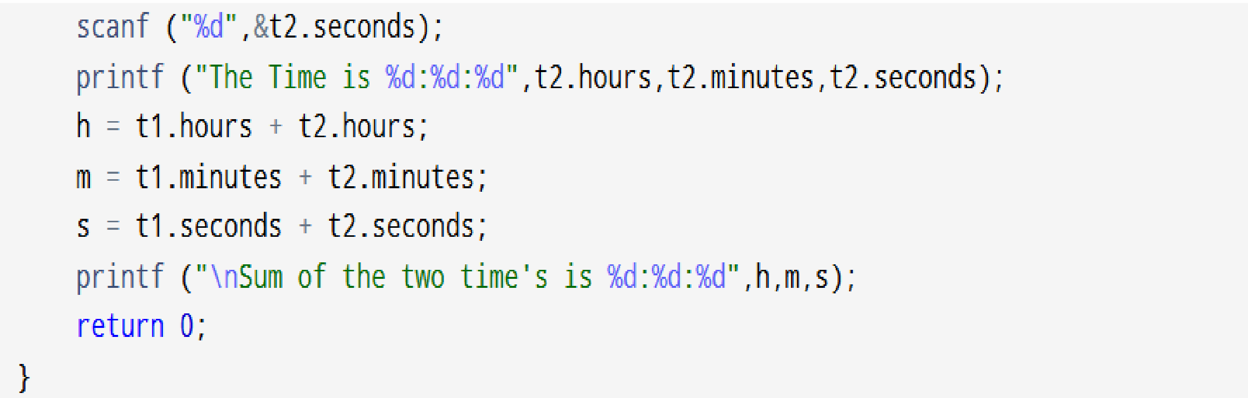
Example :

```
student1 -> CPI = 7.46;
```

Write a program to declare time structure and read two different time period and display sum of it.

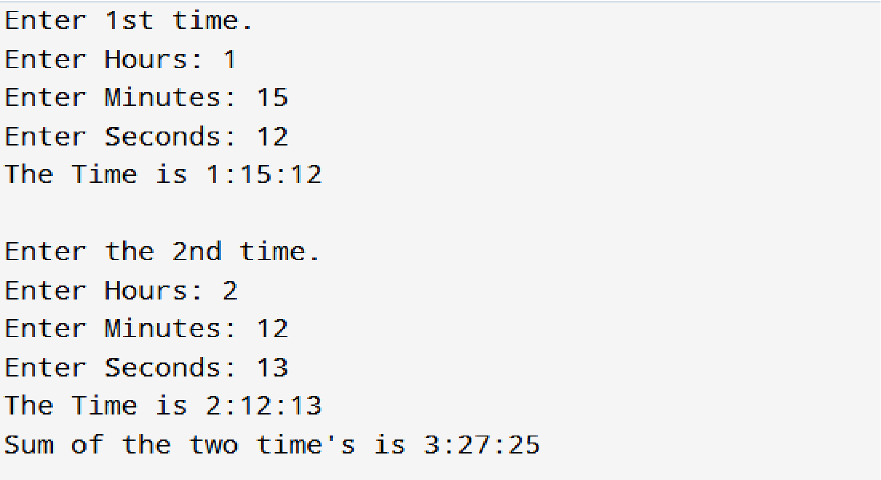


```c
#include<stdio.h>
struct time {
int hours;
int minutes;
int seconds;
};
int main()
{
    struct time t1,t2;
    int h, m, s;
    //1st time
    printf ("Enter 1st time.");
    printf ("\nEnter Hours: ");
    scanf ("%d",&t1.hours);
    printf ("Enter Minutes: ");
    scanf ("%d",&t1.minutes);
    printf ("Enter Seconds: ");
    scanf ("%d",&t1.seconds);
    printf ("The Time is %d:%d:%d",t1.hours,t1.minutes,t1.seconds);
    //2nd time
    printf ("\n\nEnter the 2nd time.");
    printf ("\nEnter Hours: ");
    scanf ("%d",&t2.hours);
    printf ("Enter Minutes: ");
    scanf ("%d",&t2.minutes);
    printf ("Enter Seconds: ");
```



```c
    scanf ("%d",&t2.seconds);
    printf ("The Time is %d:%d:%d",t2.hours,t2.minutes,t2.seconds);
    h = t1.hours + t2.hours;
    m = t1.minutes + t2.minutes;
    s = t1.seconds + t2.seconds;
    printf ("\nSum of the two time's is %d:%d:%d",h,m,s);
    return 0;
}
```

Output :



```
Enter 1st time.
Enter Hours: 1
Enter Minutes: 15
Enter Seconds: 12
The Time is 1:15:12

Enter the 2nd time.
Enter Hours: 2
Enter Minutes: 12
Enter Seconds: 13
The Time is 2:12:13
Sum of the two time's is 3:27:25
```

Structure Exercises

Exercise 1: Distance Between Two Points

Create a structure called Point that stores the coordinates of a point:

- x (integer)
- y (integer)

Write a program that:

- Accepts two points from the user.
- Calculates the Euclidean distance between the two points.

Exercise 2: Book Details

Create a structure called Book with the following members:

- Title (string)
- Author (string)
- Price (float)

Write a program that:

- Accepts the book's title, author, and price from the user.
- Displays the details of the book.

Structure and Function

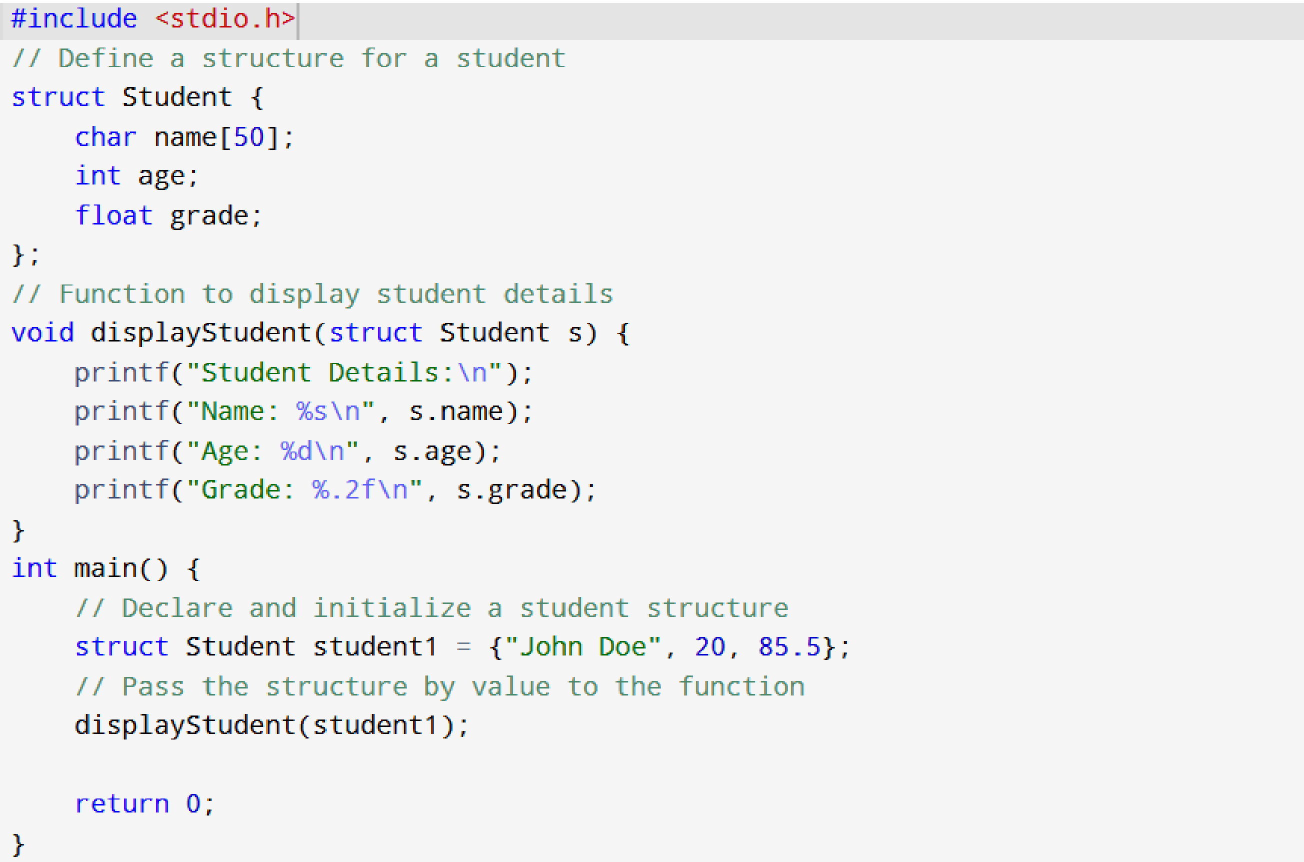
- In C programming, you can use structures in combination with functions to organize and manipulate data.
- Structures allow you to group related data together, while functions help to encapsulate operations that manipulate that data.
- When combined, structures and functions offer a powerful way to model and manage complex data.

Structure and Function

- Structures can be passed to functions either by value or reference (using pointers).

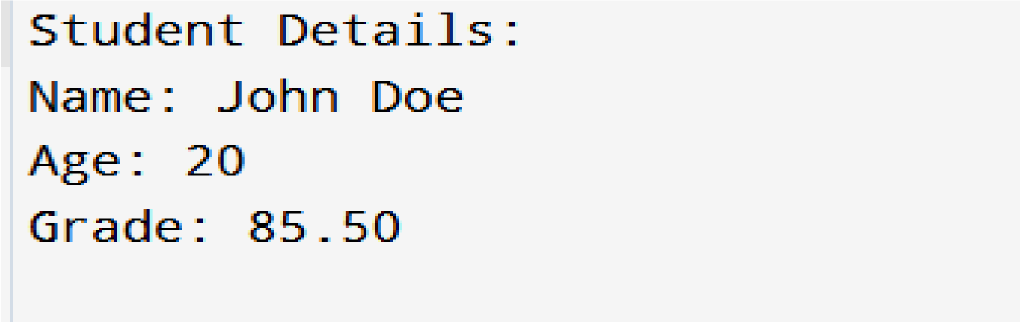
- By Value: When a structure is passed by value, the function gets a copy of the structure. Modifications made to the structure within the function will not affect the original structure.
- By Reference (Using Pointers): When a structure is passed by reference (via a pointer), the function can modify the original structure.

Structure and Function (Passing by Value)



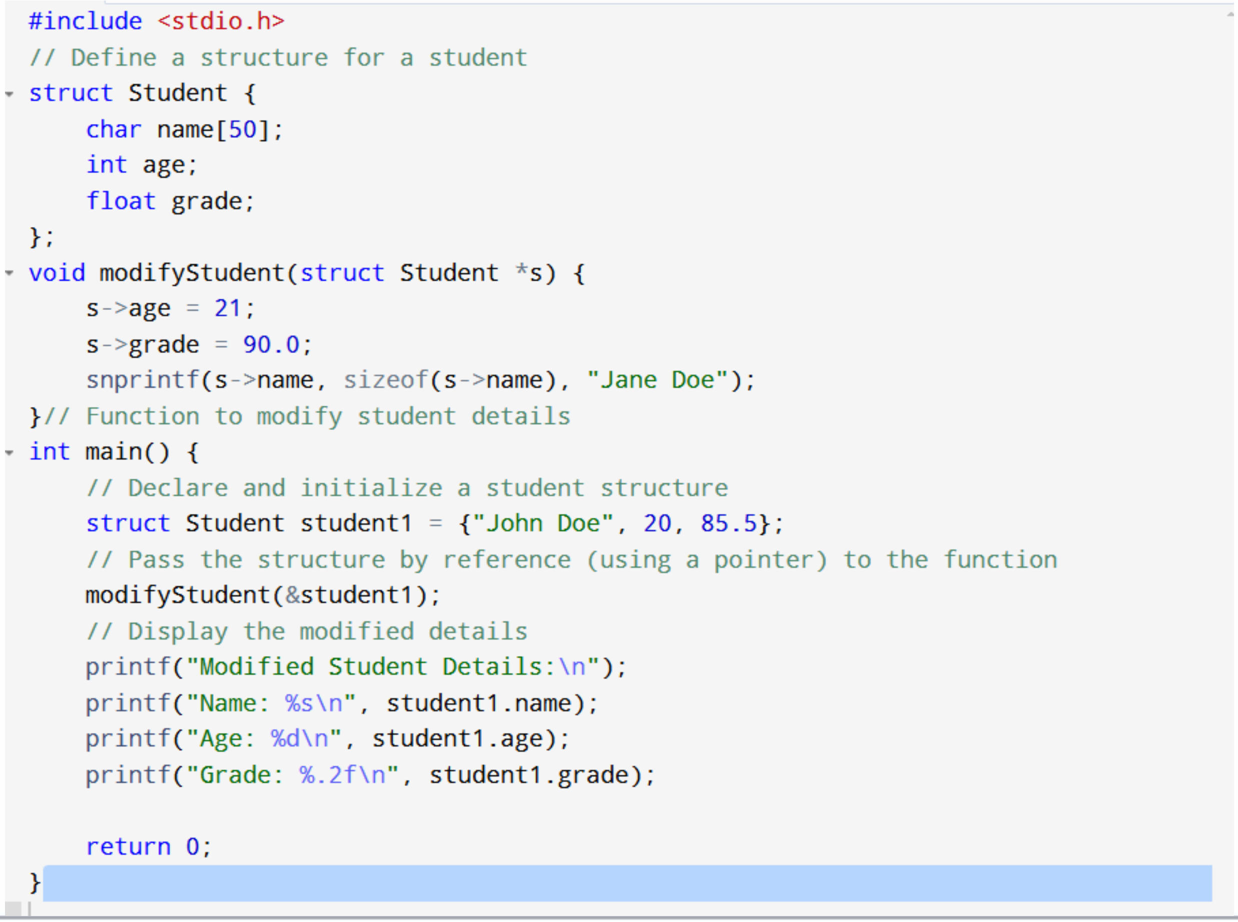
```c
#include <stdio.h>
// Define a structure for a student
struct Student {
    char name[50];
    int age;
    float grade;
};
// Function to display student details
void displayStudent(struct Student s) {
    printf("Student Details:\n");
    printf("Name: %s\n", s.name);
    printf("Age: %d\n", s.age);
    printf("Grade: %.2f\n", s.grade);
}
int main() {
    // Declare and initialize a student structure
    struct Student student1 = {"John Doe", 20, 85.5};
    // Pass the structure by value to the function
    displayStudent(student1);

    return 0;
}
```

Output :



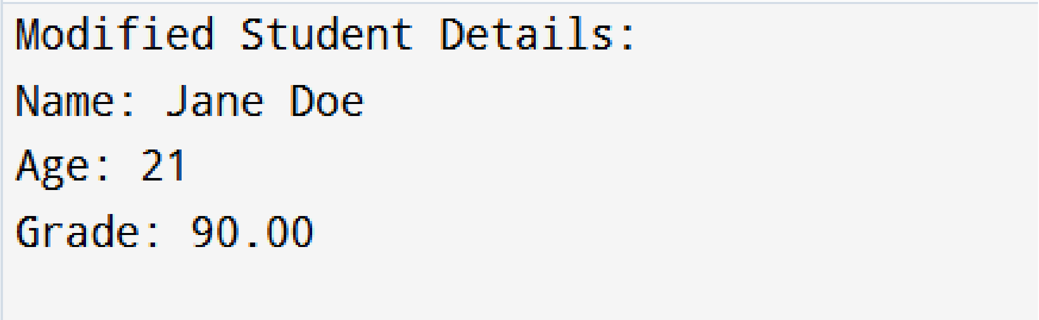
```
Student Details:
Name: John Doe
Age: 20
Grade: 85.50
```

Structure and Function (Passing by Reference)



```
#include <stdio.h>
// Define a structure for a student
struct Student {
    char name[50];
    int age;
    float grade;
};
void modifyStudent(struct Student *s) {
    s->age = 21;
    s->grade = 90.0;
    snprintf(s->name, sizeof(s->name), "Jane Doe");
}// Function to modify student details
int main() {
    // Declare and initialize a student structure
    struct Student student1 = {"John Doe", 20, 85.5};
    // Pass the structure by reference (using a pointer) to the function
    modifyStudent(&student1);
    // Display the modified details
    printf("Modified Student Details:\n");
    printf("Name: %s\n", student1.name);
    printf("Age: %d\n", student1.age);
    printf("Grade: %.2f\n", student1.grade);

    return 0;
}
```

Output :



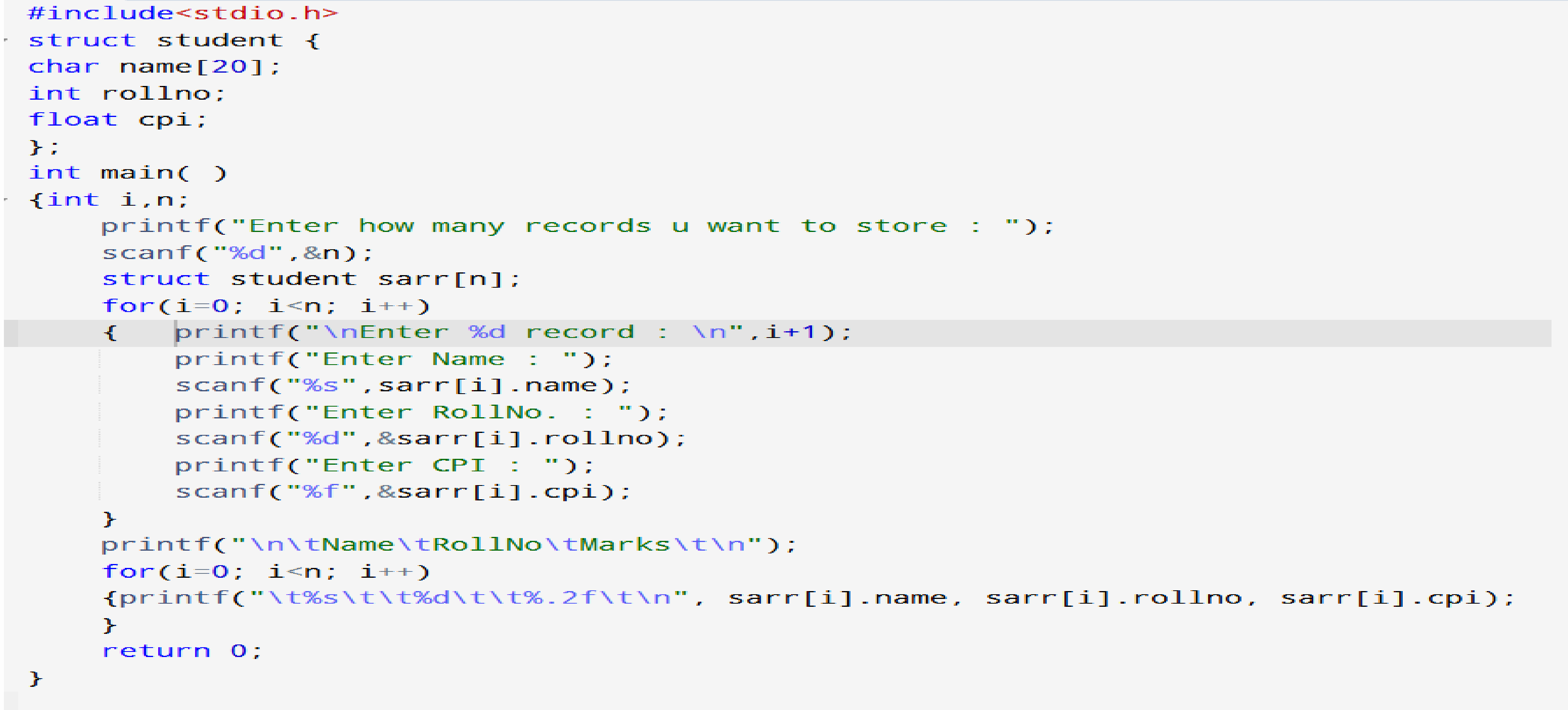
```
Modified Student Details:
Name: Jane Doe
Age: 21
Grade: 90.00
```

Array of Structure

- It can be defined as the collection of multiple structure variables where each variable contains information about different entities.
- The array of structures in C are used to store information about multiple entities of different data types.
- **Syntax :**

```
struct structure_name
{
        member1_declaration;
        member2_declaration;
        ...
        memberN_declaration;
} structure_variable[size];
```

Write a program to read and display N student information using array of structure.



```
#include<stdio.h>
struct student {
char name[20];
int rollno;
float cpi;
};
int main( )
{int i,n;
    printf("Enter how many records u want to store : ");
    scanf("%d",&n);
    struct student sarr[n];
    for(i=0; i<n; i++)
    {   printf("\nEnter %d record : \n",i+1);
        printf("Enter Name : ");
        scanf("%s",sarr[i].name);
        printf("Enter RollNo. : ");
        scanf("%d",&sarr[i].rollno);
        printf("Enter CPI : ");
        scanf("%f",&sarr[i].cpi);
    }
    printf("\n\tName\tRollNo\tMarks\t\n");
    for(i=0; i<n; i++)
    {printf("\t%s\t\t%d\t\t%.2f\t\n", sarr[i].name, sarr[i].rollno, sarr[i].cpi);
    }
    return 0;
}
```

Anonymous structure

- An anonymous structure in C is a structure that is defined without a name.
- These are typically used when you don't need to reference the structure type elsewhere in your program or when you want to create a temporary structure on the fly.
- Anonymous structures can be especially useful in situations where you need to define and use a structure for a single purpose, without needing to explicitly name the type.

Anonymous Structure

Syntax :

```
struct {
   // Structure members
} variable_name;
```

- The structure does not have a name in this case (i.e., it's anonymous).
- You can use the structure directly without needing a tag (structure name).
- You assign a variable name to this structure to access its members.

Arrays within Structures

- you can define arrays within structures to store multiple values of the same type as part of a structure.
- This allows you to group related data together in a more organized manner.
- For example, you might use an array within a structure to store multiple subjects' marks for a student or multiple measurements in a scientific application.

Syntax :

```
struct structure_name {
   data_type array_name[array_size];
};
```

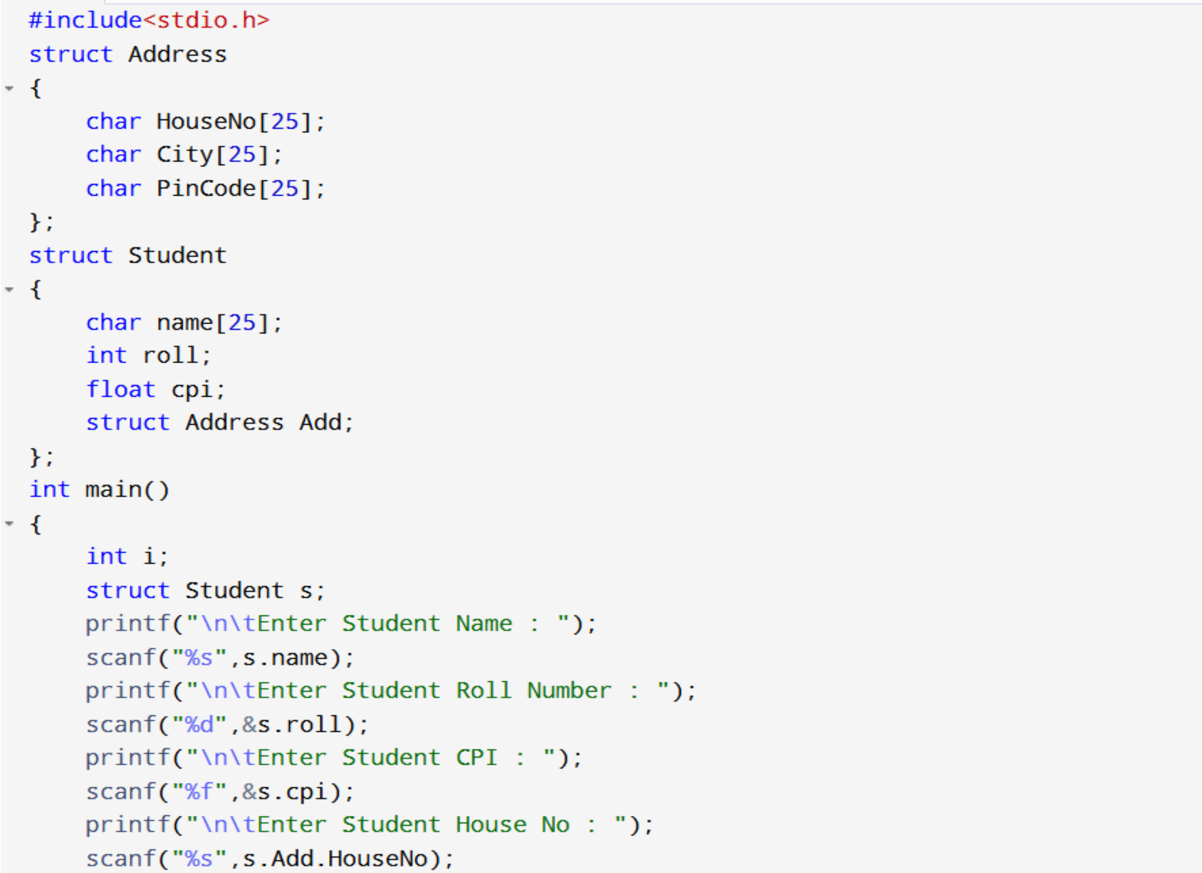
Nested Structure

- When a structure contains another structure, it is called nested structure.
- For example, we have two structures named Address and Student.
- To make Address nested to Student, we have to define Address structure before and outside Student structure and create an object of Address structure inside Student structure.

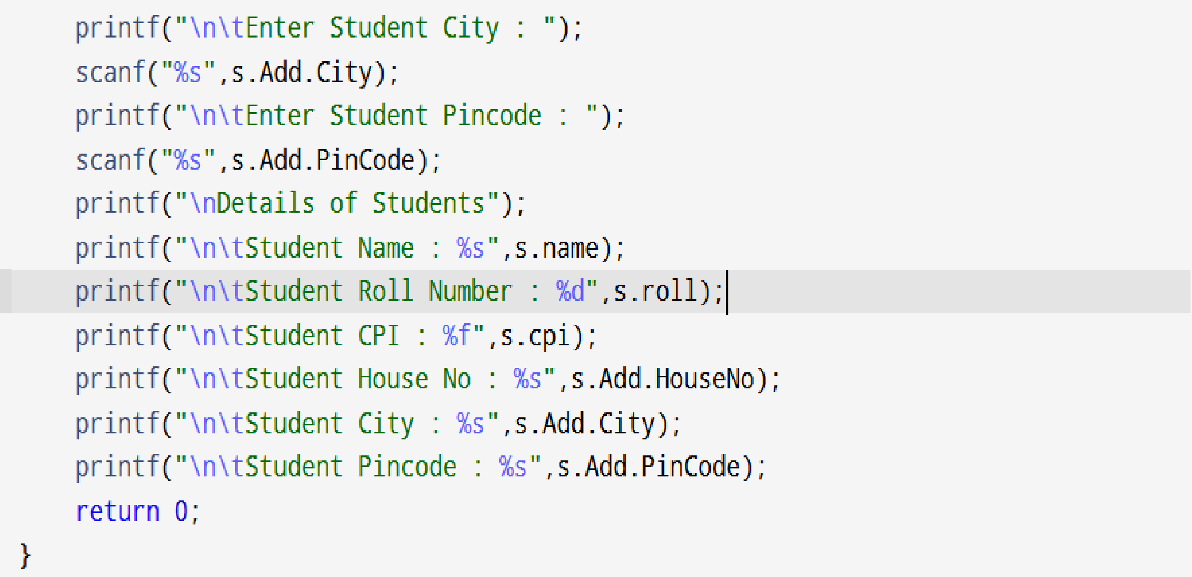
Syntax :

```
struct structure_name1
{
member1_declaration;
member2_declaration;
...
memberN_declaration;
};
struct structure_name2
{
member1_declaration;
member2_declaration;
...
struct structure1 obj;
};
```

Write a program to read and display student information using nested of structure.



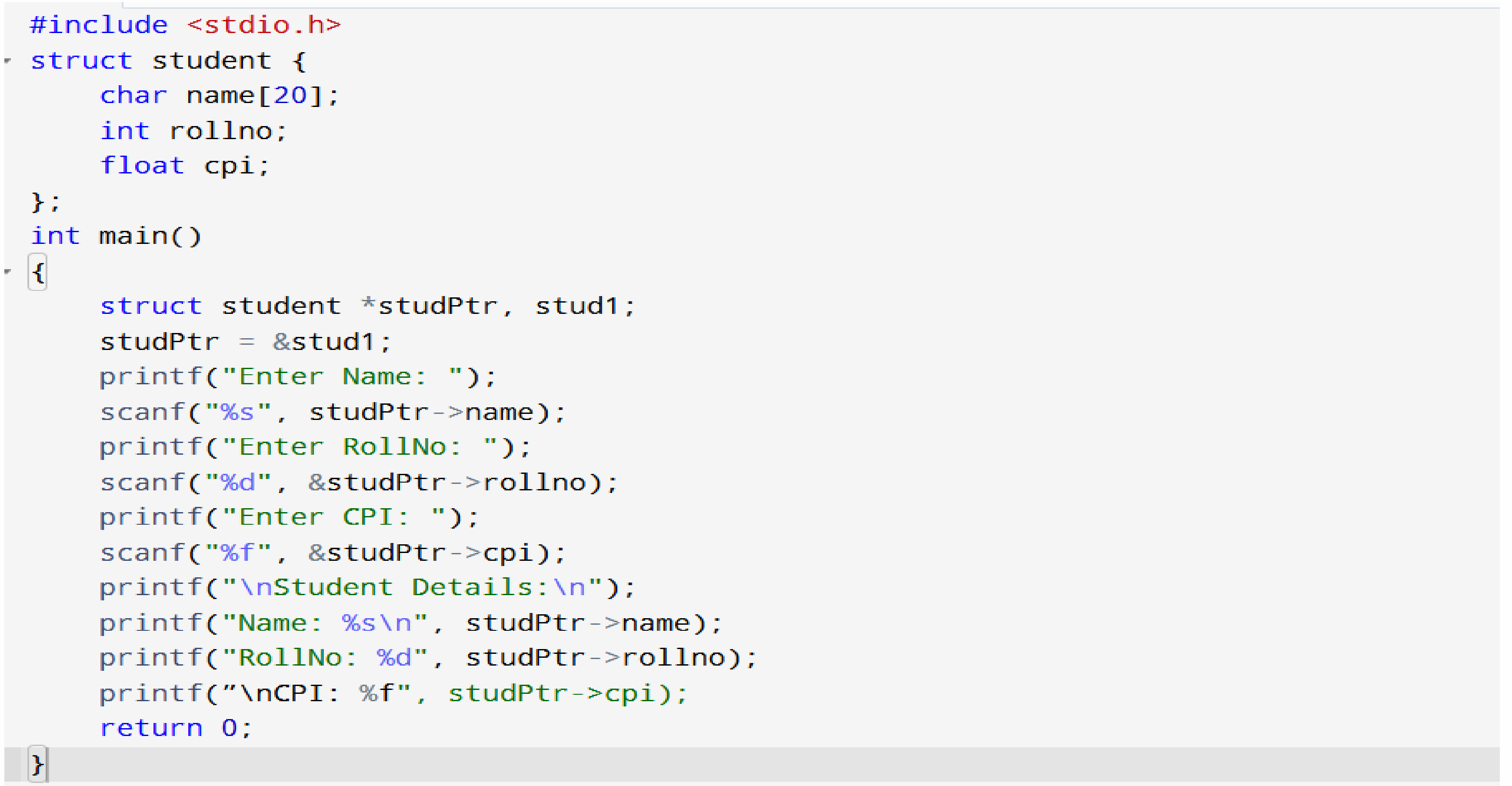
```c
#include<stdio.h>
struct Address
{
    char HouseNo[25];
    char City[25];
    char PinCode[25];
};
struct Student
{
    char name[25];
    int roll;
    float cpi;
    struct Address Add;
};
int main()
{
    int i;
    struct Student s;
    printf("\n\tEnter Student Name : ");
    scanf("%s",s.name);
    printf("\n\tEnter Student Roll Number : ");
    scanf("%d",&s.roll);
    printf("\n\tEnter Student CPI : ");
    scanf("%f",&s.cpi);
    printf("\n\tEnter Student House No : ");
    scanf("%s",s.Add.HouseNo);
```



```c
    printf("\n\tEnter Student City : ");
    scanf("%s",s.Add.City);
    printf("\n\tEnter Student Pincode : ");
    scanf("%s",s.Add.PinCode);
    printf("\nDetails of Students");
    printf("\n\tStudent Name : %s",s.name);
    printf("\n\tStudent Roll Number : %d",s.roll);
    printf("\n\tStudent CPI : %f",s.cpi);
    printf("\n\tStudent House No : %s",s.Add.HouseNo);
    printf("\n\tStudent City : %s",s.Add.City);
    printf("\n\tStudent Pincode : %s",s.Add.PinCode);
    return 0;
}
```

Pointer to Structure

- Reference/address of structure object is passed as function argument to the definition of function.



```
#include <stdio.h>
struct student {
    char name[20];
    int rollno;
    float cpi;
};
int main()
{
    struct student *studPtr, stud1;
    studPtr = &stud1;
    printf("Enter Name: ");
    scanf("%s", studPtr->name);
    printf("Enter RollNo: ");
    scanf("%d", &studPtr->rollno);
    printf("Enter CPI: ");
    scanf("%f", &studPtr->cpi);
    printf("\nStudent Details:\n");
    printf("Name: %s\n", studPtr->name);
    printf("RollNo: %d", studPtr->rollno);
    printf(”\nCPI: %f", studPtr->cpi);
    return 0;
}
```

Self-referential structure

- A self-referential structure in C is a structure that contains a pointer to the same type of structure.
- This is useful for creating linked lists, trees, or other recursive data structures, where each element needs to point to another element of the same type.

Structure Padding

- Structure padding refers to the technique used by compilers to align the members of a structure in memory to improve access speed and prevent potential performance issues.
- It involves adding extra memory (called "padding") between the members of a structure, ensuring that each member is stored at an appropriate memory address that adheres to the architecture's alignment rules.
- The size of a structure can be larger than the sum of its members due to padding.
- You can control structure padding using compiler-specific directives (like #pragma pack), but be careful, as this might affect performance or cause issues on certain platforms.

Union

- Union is a user defined data type similar like Structure.
- It holds different data types in the same memory location.
- You can define a union with various members, but only one member can hold avalue at any given time.
- Union provide an efficient way of using the same memory location for multiple-purpose.

Syntax to Define and Access Union

- Declaration of union must start with the keyword union followed by the union name and union’s member variables are declared within braces.
- **Syntax :**

```
union union_name
{
        member1_declaration;
        member2_declaration;
        . . .

        memberN_declaration;
};
```

Accessing the union members

- You need to create an object of union to access its members.
- Object is a variable of type union. Union members are accessed using the dot operator(.) between union's object and union's member name.
- **Syntax :**

  union union_name union_variable;
- You must terminate union definition with semicolon ;.
- You cannot assign value to members inside the union definition, it will cause compilation error.

Example to Define Union

Example :

```
union student
{
    char name[30]; // Student Name
    int roll_no; // Student Roll No
    float CPI; // Student CPI
    int backlog; // Student Backlog
} student1;
```

Key Characteristics of Union:

- Memory Sharing
- Only One Member at a Time
- Size of Union
- Use Cases

Structure vs. Union

Feature	Structure	Union
Memory Allocation	Each member has its own memory space	All members share the same memory
Size	Sum of the sizes of all members	Size of the largest member
Access to Members	All members can hold valid values	Only one member can hold a value at a time
Use Case	Storing multiple values simultaneously	Storing one value at a time, but in different formats
Overwriting	No overwriting of data	Storing a value overwrites other values
Example Use	Student record with ID, name, GPA	Data that can be an int, float, or char[] but only one at a time

Thank You