**🧠 Unit: Pointers, Structures, and Unions in C**

**🔹 1. Pointers**

**Definition**

A **pointer** is a variable that stores the **memory address** of another variable.  
In C, pointers are used to access and manipulate data stored in memory directly.

**Syntax**

data\_type \*pointer\_name;

**Example**

int \*p; // pointer to integer

float \*f; // pointer to float

char \*c; // pointer to character

**Simple Example of Pointer**

#include <stdio.h>

int main() {

int a = 10; // normal variable

int \*p; // pointer variable declaration

p = &a; // storing address of a in pointer p

printf("Value of a = %d\n", a);

printf("Address of a = %p\n", &a);

printf("Value of pointer p = %p\n", p);

printf("Value at address pointed by p = %d\n", \*p);

return 0;

}

**Output**

Value of a = 10

Address of a = 0x7ffd12ab

Value of pointer p = 0x7ffd12ab

Value at address pointed by p = 10

🧩 **Explanation:**

* &a → gives address of variable a.
* p → stores that address.
* \*p → gives the value stored at that address.

**Pointer and Array Example**

#include <stdio.h>

int main() {

int arr[5] = {10, 20, 30, 40, 50};

int \*p = arr; // arr itself acts as pointer

printf("Array elements using pointer:\n");

for (int i = 0; i < 5; i++) {

printf("%d ", \*(p + i));

}

return 0;

}

**Output**

Array elements using pointer:

10 20 30 40 50

🧩 **Explanation:**  
\*(p + i) accesses array elements using pointer arithmetic.

**Pointer to Pointer Example**

#include <stdio.h>

int main() {

int a = 25;

int \*p = &a;

int \*\*q = &p;

printf("Value of a = %d\n", a);

printf("Value using pointer p = %d\n", \*p);

printf("Value using pointer to pointer q = %d\n", \*\*q);

return 0;

}

**Output**

Value of a = 25

Value using pointer p = 25

Value using pointer to pointer q = 25

**Function Pointer Example**

#include <stdio.h>

void greet() {

printf("Hello, Welcome to C Programming!\n");

}

int main() {

void (\*ptr)(); // function pointer declaration

ptr = greet; // assigning function address

ptr(); // calling function using pointer

return 0;

}

**Output**

Hello, Welcome to C Programming!

**🔹 2. Structures**

**Definition**

A **structure** is a user-defined data type that groups variables of **different data types** under one name.  
It helps organize complex data, like a student record containing name, roll number, and marks.

**Syntax**

struct structure\_name {

data\_type member1;

data\_type member2;

...

};

**Simple Structure Example**

#include <stdio.h>

#include <string.h>

struct Student {

int id;

char name[20];

float marks;

};

int main() {

struct Student s1;

s1.id = 101;

strcpy(s1.name, "Ravi");

s1.marks = 89.5;

printf("Student ID: %d\n", s1.id);

printf("Student Name: %s\n", s1.name);

printf("Student Marks: %.2f\n", s1.marks);

return 0;

}

**Output**

Student ID: 101

Student Name: Ravi

Student Marks: 89.50

**Array of Structure Example**

#include <stdio.h>

#include <string.h>

struct Student {

int id;

char name[20];

float marks;

};

int main() {

struct Student s[3];

for (int i = 0; i < 3; i++) {

printf("Enter ID, Name, Marks for student %d: ", i + 1);

scanf("%d %s %f", &s[i].id, s[i].name, &s[i].marks);

}

printf("\nStudent Details:\n");

for (int i = 0; i < 3; i++) {

printf("ID: %d, Name: %s, Marks: %.2f\n", s[i].id, s[i].name, s[i].marks);

}

return 0;

}

**Output (Example Input)**

Enter ID, Name, Marks for student 1: 101 Ravi 88

Enter ID, Name, Marks for student 2: 102 Meera 92

Enter ID, Name, Marks for student 3: 103 Amit 85

Student Details:

ID: 101, Name: Ravi, Marks: 88.00

ID: 102, Name: Meera, Marks: 92.00

ID: 103, Name: Amit, Marks: 85.00

**Structure with Pointer Example**

#include <stdio.h>

struct Student {

int id;

float marks;

};

int main() {

struct Student s1 = {104, 95.0};

struct Student \*ptr = &s1;

printf("Student ID: %d\n", ptr->id);

printf("Student Marks: %.2f\n", ptr->marks);

return 0;

}

**Output**

Student ID: 104

Student Marks: 95.00

🧩 **Explanation:**  
The arrow operator -> is used with structure pointers to access members.

**🔹 3. Unions**

**Definition**

A **union** is similar to a structure, but all members share the **same memory location**.  
Only one member can be used at a time — this saves memory.

**Syntax**

union union\_name {

data\_type member1;

data\_type member2;

...

};

**Example**

#include <stdio.h>

union Data {

int i;

float f;

char ch;

};

int main() {

union Data d;

d.i = 10;

printf("Integer: %d\n", d.i);

d.f = 25.5;

printf("Float: %.2f\n", d.f);

d.ch = 'A';

printf("Character: %c\n", d.ch);

return 0;

}

**Output**

Integer: 10

Float: 25.50

Character: A