Advanced Pointers and Structs

Pointer Typecasting, Pointer Arithmetic, Pointers to Pointers, and Arrays of Pointers

Pointer Typecasting

Example 1: Casting `int*` to `char*`

```c
#include <stdio.h>

int main() {
    int x = 0x12345678;
    char *p = (char*)&x; // cast int pointer to char pointer

    printf("x = 0x%x\n", x);
    printf("First byte = 0x%x\n", *p);
    return 0;
}
```

```c
#include <stdio.h>

int main() {
    double val = 3.1415926535;
    unsigned char *p = (unsigned char*)&val;

    printf("Memory of double:\n");
    for (int i = 0; i < sizeof(double); i++)
        printf("%02x ", p[i]);
}
```

Pointer Arithmetic Recap

```
int arr[5] = {10, 20, 30, 40, 50};
int *p = arr;
printf("%d\n", *(p + 2)); // 30
```

🔷 Array of Pointers

Example 1: Strings (Array of Char Pointers)

```c
#include <stdio.h>

int main() {
    const char *names[] = {"Alice", "Bob", "Charlie"};
    for (int i = 0; i < 3; i++)
        printf("%s\n", names[i]);
    return 0;
}
```

💡 Each element names[i] is a char* pointing to a string literal.

Structs

🧩 1. Definition of a Struct

A **structure (struct)** in C is a user-defined data type that allows grouping variables of different types under a single name.

```c
// Example: Defining a struct for a student
struct Student {
    int roll_no;
    char name[50];
    float marks;
};
```

Notes:

- A struct groups logically related data.
- Members can be of **different data types**.

he keyword `struct` is used to declare a structure.

🏗️ 2. Declaring and Initializing Struct Variables

Method 1: Separate Declaration

```
struct Student s1;
s1.roll_no = 101;
strcpy(s1.name, "Alice");
s1.marks = 89.5;
```

Method 2: Initialization at Declaration

```
struct Student s2 = {102, "Bob", 92.0};
```

Method 3: Designated Initializers (C99 and later)

```
struct Student s3 = {
    .roll_no = 103,
    .name = "Charlie",
    .marks = 95.2
```

🧭 3. Accessing Struct Members

You can access structure members using the **dot operator (.)**.

```
printf("Roll No: %d\n", s1.roll_no);
printf("Name: %s\n", s1.name);
printf("Marks: %.2f\n", s1.marks);
```

💡 4. Structs with Pointers

When using pointers to structs, use the **arrow operator (->)**.

```
struct Student *ptr = &s2;
printf("Name (via pointer): %s\n", ptr->name);
ptr->marks = 93.5;
```

⚙ 5. Array of Structs

You can create an array of structs to store multiple records.

```
struct Student class[3] = {
    {101, "Alice", 89.5},
    {102, "Bob", 92.0},
    {103, "Charlie", 95.2}
};
```

Access Example:

```
for(int i = 0; i < 3; i++) {
    printf("%d %s %.2f\n", class[i].roll_no, class[i].name, class[i].marks);
}
```

🔄 6. Passing Structs to Functions

Pass by Value

```
void printStudent(struct Student s) {
    printf("%d %s %.2f\n", s.roll_no, s.name, s.marks);
}
```

Pass by Reference

```
void updateMarks(struct Student *s, float newMarks) {
    s->marks = newMarks;
}
```

🧠 7. Nested Structs

Structs can contain other structs as members.

```
struct Date {
    int day, month, year;
};

struct Student {
    int roll_no;
    char name[50];
    struct Date dob;
};
```

Access Example:

```
struct Student s = {101, "Alice", {12, 5, 2003}};
    tf("DOB: %d/%d/%d\n", s.dob.day, s.dob.month, s.dob.year);
```

```c
#include "stdio.h"
// For using strcpy, strlen functions
#include "string.h"

// Defining a structure to store the details of a single student
struct Student {
        char name[100];
        char email[100];
        float marks;
};


int main() {

  // create a array of struct Student to store data for the whole class
  // here only 2 students are there as an example
        struct Student class[2] = {
                { .name = "Raju", .marks = 25.5 }, // initializer syntax
                { "Ammu", "ammu@research.iiit.ac.in", 45.0 } // initiallizer syntax
        };
        strcpy(class[0].name, "Girish Varma");

        class[0].marks = 80.0;

        printf("%lu\n", sizeof(struct Student));

         printf("Student Details:\nName:\t %s %lu \nEmail:\t %s\nMarks:\t %f\n", class[0].name, strlen(class[0].name), class[0].email, class[0].marks);
        printf("Student Details:\nName:\t %s\nEmail:\t %s\nMarks:\t %f\n", class[1].name, class[1].email, class[1].marks);

        return 0;
}
```

```c
#include<stdio.h>
struct rectangle {
        float length;
        float breadth;
};

float compute_area(struct rectangle r) {
        return r.length * r.breadth;
}

void print_rectangle(struct rectangle r) {
        printf("Rectangle with length %f and breadth %f\n", r.length, r.breadth);
}

int main()
{
        struct rectangle rect = { 1.5, 3.2 }; // Initializer
        print_rectangle(rect);
        printf("Area of the rectangle is %f \n", compute_area(rect) );
}
```

Struct with Typedef

```c
#include<stdio.h>
typedef struct rectangle {
        float length;
        float breadth;
} rectangle;

float compute_area(rectangle r) {
        return r.length*r.breadth;
}

rectangle scale(rectangle r, float s) {
        r.length = r.length*s;
        r.breadth = r.breadth*s;
        return r;
}

int main()
{
        rectangle rect = {  .breadth = 1.0, .length = 3.0} /* {3.0, 1.0 }*/;
        // rect.length = 3.2;
        // rect.breadth = 1.2;
        printf("Area of the rectangle is %f \n", compute_area(rect));
        rectangle rp = scale(rect, 5);
        printf("Area of the rectangle is %f \n", compute_area(rp));
        printf("Area of the rectangle is %f \n", compute_area(rect));
}
```

Passing using pointers

```c
#include<stdio.h>
typedef struct rectangle {
        float length;
        float breadth;
} rectangle;

float compute_area(rectangle r) {
        return r.length*r.breadth;
}

rectangle* scale(rectangle* r, float s) {
        r->length = r->length*s;
        r->breadth = r->breadth*s;
        return r;
}

int main()
{
        rectangle rect = {  .breadth = 1.0, .length = 3.0} /* {3.0, 1.0 }*/;
        // rect.length = 3.2;
        // rect.breadth = 1.2;
        printf("Area of the rectangle is %f \n", compute_area(rect));
        rectangle* rp = scale(&rect, 5);
        printf("Area of the rectangle is %f \n", compute_area(*rp));
        printf("Area of the rectangle is %f \n", compute_area(rect));
}
```