Defining and Processing a Structure, User Defined Data Types in C Programming

(1) Structure in C

- A structure is a user-defined data type in C that allows us to group different data types under a single name.
- It is used to represent a real-world object (like a student, employee, etc.).
- A structure contains variables of different data types.

Defining a Structure

The keyword struct is used to define a structure.

Syntax:

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

Example:

```
struct student {
    int roll_no;
    char name[50];
    float marks;
};
```

Note:

- struct is a keyword.
- structure_name is the name of the structure.
- The variables inside the braces are called members.

(2) Processing a Structure

- After defining a structure, we can create variables of that structure type.
- We can access the members using the dot (.) operator.
- We can also initialize values, take input and display output.

(a) Declaration and Initialization

Example:

```
struct student s1 = {101, "Kartik", 85.5};   // Initializes all members
```

(b) Accessing Members

Example:

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

// Use dot (.) operator to access members of a structure.

(c) Taking Input

Example:

```
printf("Enter roll no: ");
scanf("%d", &s1.roll_no);
printf("Enter name: ");
scanf("%s", s1.name);
printf("Enter marks: ");
scanf("%f", &s1.marks);
```

// Use & operator for input (except for arrays).

Note:

- While using scanf() for character arrays (like name), use %s.
- For arrays, you don't need & symbol, but for other variables you should use &.

(d) Displaying Output

Example:

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

(3) User Defined Data Types in C

- In C, we can create our own data types using structures, typedef and enum.
- These are called user-defined data types because the programmer defines them.

Example using typedef:

```
typedef struct {
    int id;
    char name[30];
    float salary;
} Employee;
```

// Now Employee is a new data type.
// We can declare variables like:
Employee e1;

Other User Defined Data Types:

- typedef → to create an alias for existing data types.
- enum → to define a set of named constants.
- struct → to create a structure.

In short:

Structure → groups different data types, Processing → define, initialize, access members,
User defined data types → struct, typedef, enum.

Structures and Pointers in C Programming

1 Structures in C

- A structure is a user-defined data type in C
- that allows us to group different data types under a single name.
- It is used to represent a real-world object like a student, product, employee, etc.
- A structure contains variables of different data types.

Syntax :-

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

Example :- Define and use a structure

```
struct student {
    int roll_no;
    char name[50];
    float marks;
};
void main() {
  struct student s1;
  s1.roll_no = 101;
  strcpy(s1.name, "Kartik");
  s1.marks = 85.5;
  printf("Roll No: %d\n", s1.roll_no);
  printf("Name: %s\n", s1.name);
  printf("Marks: %.2f\n", s1.marks);
}
```

Note :-

- Use dot (.) operator to access members of a structure.
- The members are stored in contiguous memory locations.

Accessing Structure Members using Pointer :-

We can also access structure members using a pointer with the help of -> operator.

Example :-

```
struct student *p;
p = &s1;
printf("Name: %s\n", p->name);
printf("Marks: %.2f\n", p->marks);
```

Note :-

- The -> operator is used to access members through a pointer.

2 Pointers in C

- A pointer is a variable that stores the address of another variable.
- It is used to access and manipulate data indirectly.
- Pointers help in dynamic memory allocation, function calling, and efficient programming.

Syntax :-

```
data_type *pointer_name;
```

Example :-

```
int a = 10;
int *p;
p = &a;     // p now holds the address of a
```

Accessing Value using Pointer :-

- To get the value stored at the address, we use the dereference operator (*).

Example :-

```
int a = 10;
int *p = &a;
printf("Value of a = %d\n", *p);    // prints 10
```

Pointer to Structure :-

- We can use pointers to access structure members using -> operator.

Example :-

```
Struct student *p = &s1;
printf("Name: %s\n", p->name);
```

Note :-

- Pointer stores the memory address.
- * (dereference) gives the value at that address.
- & (address-of) gives the address of a variable.
- Pointers are very useful with structures, arrays, functions and dynamic memory.

3 Key Points / Summary

Structures :-

- Group different data types.
- Used for real-world representation.
- Access using . (dot) operator.
- Can be used with pointers using -> operator.

Pointers :-

- Store memory addresses.
- Use & to get address.
- Use * to get value.
- Work with structures, arrays, functions, etc.

1. Passing Structures to Functions

In C, we can pass structures to functions in two ways:

1) Pass by Value (copy of the structure)
2) Pass by Address (using pointer)

1. Pass by Value (Copy of structure)

- When a structure is passed by value, a copy of the structure is made and passed to the function.
- Any changes made inside the function do not affect the original structure.

Example:

```
#include <stdio.h>
struct Student {
    int id;
    char name[20];
};
void display(struct Student s) {   // pass by value
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
}
int main() {
    struct Student st = {101, "Kartik"};
    display(st);     // structure passed by value
    return 0;
}
```

2. Pass by Address (Using Pointer)

- When a structure is passed by address, the address of the structure is passed to the function.
- Any changes made inside the function will affect the original structure.

Example:

```
void update(struct Student *s) {     // pass by address
    s->id = 202;
    printf("Updated ID: %d\n", s->id);
}
// In main()
 update(&st);  // address of structure passed
```

2. Unions in C Programming

What is a Union?

- A union is a special type of user-defined data type in C that can hold different data types at the same memory location.
- Unlike structures, where each member has its own memory location, in a union all members share the same memory location.
- The size of a union is equal to the size of its largest member.

Syntax:

```
union union_name {
      data_type member1;
      data_type member2;
      data_type member3;
      ...
};
```

Example:

```
#include <stdio.h>
union Data {
    int i;
    float f;
    char str[10];
};
int main() {
    union Data d;
    d.i = 10;
    printf("Integer: %d\n", d.i);
    d.f = 5.5;    // now float is stored, previous value
                     is lost
    printf("Float: %f\n", d.f);
    d.str[0] = 'A';
    printf("Char: %c\n", d.str[0]);
    printf("Size of union: %lu\n", sizeof(union Data));
    return 0;
}
```

Key Points:

- Only one member can be used at a time.
- Changing one member affects the others.
- Unions are useful for memory optimization and for handling different data types in the same memory location.