Session 09 C

Thursday, 6 November 2025 11:04 AM

→ Type Conversion / Type Casting.

→ Structure / Union

→ Break & Continue

→ Sorting → Bubble Sort / Selection Sort.

→

① Micro

② Linked list

→ calloc malloc

→ Dom

→ file handling.

🔷 Type Conversion

Type conversion means **changing the data type of a variable automatically or manually**.
There are two types of type conversion:

1. Implicit Type Conversion (Type Promotion / Automatic Conversion)

- Done **automatically by the compiler**.
- Also called **Type Promotion**.
- Happens when you perform operations between two different data types.
- The **smaller data type is promoted to a larger data type** to prevent data loss.

📘 **Example:**

```
#include <stdio.h>
int main() {
   int a = 5;
   float b = 2.5;
   float result = a + b; // int 'a' is automatically converted to float
   printf("Result = %f", result);
   return 0;
}
```

Output:

```
Result = 7.500000
```

Here, a (int) is automatically converted to float before addition.

📈 **Type Conversion Hierarchy (Lower → Higher)**

char → int → float → double

2. Explicit Type Conversion (Type Casting)

- Done **manually by the programmer**.
- Use a **cast operator (data_type)** before the variable.
- Gives more **control** over how conversions are made.

📘 **Example:**

```
#include <stdio.h>
int main() {
   int a = 10, b = 3;
   float result;
result = (float)a / b;  // Explicitly cast 'a' to float
   printf("Result = %.2f", result);
   return 0;
}
```

Output:

```
Result = 3.33
```

Here, (float)a converts a from int to float, so division is done in floating point.

🧠 Difference Between Type Conversion and Type Casting

Feature	Type Conversion	Type Casting
Who Performs	Compiler automatically	Programmer manually
Also Known As	Implicit Conversion / Type Promotion	Explicit Conversion
Control	No control by programmer	Full control by programmer
Syntax	Happens automatically	(data_type) expression
Example	int + float → float	(float)a / b

Difference Between Break & Continue

```
for( int i=0; i<10; i++)
{
    printf("The val of i=%d", i);
    if( i==5) break;
    printf("Hello");
}
```

Continue.

i=0
Hello
i=1
Hello
i=2
Hello
i=3
Hello
i=4
Hello
i=5
i=6

The val of i=0
Hello
The val of i=1
Hello
The val of i=2
Hello
The val of i=3
Hello
i=4
Hello
i=5

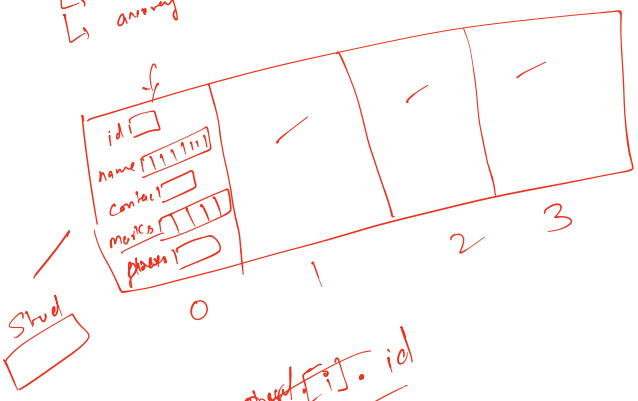
```
int arr[10];
char arr[10];
```

Student Record
- id — int
- name = char arr[];
- contact — int
- marks[5]; — float
- average — float

```
struct Student{
    int id;
    char name[30];
    int contact;
    float marks[5];
    float average;
};
```

4
30
4
20
4
64

struct Student stud[4];



&stud[i].id
~~stud.name~~ &stud[i].name
&stud.contact
&stud[i].marks[0]
&std[i].avg.

~~&stud[i].id~~

Structure in C

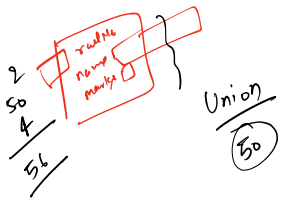
A **structure** in C is a **user-defined data type** that allows you to combine **different types of data** (like int, float, char, etc.) into a **single unit**.
It is mainly used to represent a **record** — for example, a student, employee, or product with multiple attributes.

Syntax of Structure

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

Example:

```
#include <stdio.h>
#include <string.h>
struct Student {
   int roll_no;
   char name[50];
   float marks;
};
int main() {
   // Declare a structure variable
   struct Student s1;
// Assign values to members
   s1.roll_no = 101;
   strcpy(s1.name, "Rahul");
   s1.marks = 89.5;
// Display data
   printf("Roll No: %d\n", s1.roll_no);
   printf("Name: %s\n", s1.name);
   printf("Marks: %.2f\n", s1.marks);
return 0;
}
```



Output:

```
Roll No: 101
Name: Rahul
Marks: 89.50
```

Key Points About Structures

1. **User-defined:** Unlike arrays (same data type), structures can hold **different data types**.
2. **Members are accessed using dot (.) operator.**

   ```
   s1.marks = 90.0;
   printf("%d", s1.roll_no);
   ```
3. **Structure variables are independent** — each has its own copy of data.
4. You can create **multiple variables** of the same structure type:

   ```
   struct Student s1, s2, s3;
   ```
5. Structures can be **passed to functions** and **returned from functions**.
6. You can also create an **array of structures**.

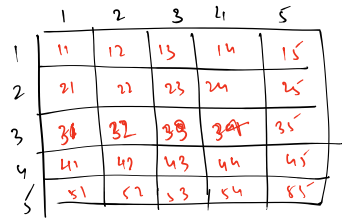
Array of Structures Example

```
#include <stdio.h>
#include <string.h>
struct Student {
   int roll_no;
   char name[50];
   float marks;
};
int main() {
   struct Student s[2];
// Input for student 1
   s[0].roll_no = 101;
   strcpy(s[0].name, "Amit");
   s[0].marks = 88.5;
// Input for student 2
   s[1].roll_no = 102;
   strcpy(s[1].name, "Neha");
   s[1].marks = 92.0;
// Display
   for (int i = 0; i < 2; i++) {
      printf("\nRoll No: %d\nName: %s\nMarks: %.2f\n", s[i].roll_no, s[i].name, s[i].marks);
   }
return 0;
}
```

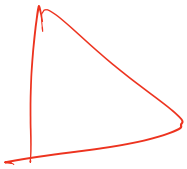
Difference Between Structure and Array

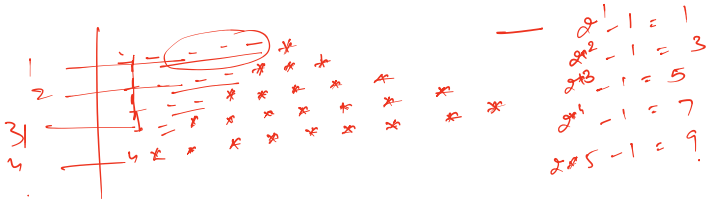
Feature	Structure	Array
Data Type	Can store different data types	Stores only one data type
Syntax	struct Student {int; char; float;}	int arr[5];
Purpose	Represents a record	Represents a collection of similar items

In Short:
A **structure** groups **different types of data** under one name — useful for representing real-world entities like students, employees, or books.




```
for( i=1;  i<=N; i++ )->row
  {
  for( j=1; j<=N; j++)->col
    {
      if( condi )
        {
          1
        // else {
        }
    }
  printf(" \n");
  }
```





i>1