Certainly! Let's break down **Question 2: Data Types in C** in detail.

---

2. Data Types in C

A **data type** in C specifies the **type of data** that a variable can store. It defines:
- The amount of **memory** allocated
- The **operations** that can be performed on the data

C has the following major types of data types:

1. Primary (Basic) Data Types

These are the fundamental data types in C.

Data Type	Size (in bytes)	Range
`char`	1 byte	-128 to 127 (signed) or 0 to 255 (unsigned)
`int`	2 or 4 bytes	-32,768 to 32,767 (2 bytes) / -2,147,483,648 to 2,147,483,647 (4 bytes)
`float`	4 bytes	6 decimal places
`double`	8 bytes	15 decimal places

Example: Declaring Basic Data Types

```c
#include <stdio.h>
int main() {
    char grade = 'A';   // Character type
    int age = 20;       // Integer type
    float pi = 3.14;    // Floating-point type
    double largePi = 3.14159265359; // Double type

    printf("Grade: %c\n", grade);
    printf("Age: %d\n", age);
    printf("Pi: %.2f\n", pi);
    printf("Large Pi: %.10lf\n", largePi);

    return 0;
}
```

---

2. Derived Data Types

Derived from primary types:

Data Type	Description
Array	Collection of elements of the same data type (e.g., `int arr[10];`)

Data Type	Description
Pointer	Stores memory address of another variable (e.g., `int *ptr;`)
Structure	Groups different data types under one name (`struct student {int id; char name[20];};`)
Union	Similar to structures but shares memory between variables

Example of Array

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

---

3. User-Defined Data Types

Created using **typedef** and **enum**.

(a) typedef (Alias for Data Type)

```
typedef unsigned int uint;  // Now 'uint' is an alias for 'unsigned int'
uint num = 10;
```

(b) enum (Enumeration)

Used for defining a set of constants.

```
enum days {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
enum days today = Monday;
```

---

4. Void Data Type

- `void` means **no value**.
- Used in **functions** that do not return any value.

```
void greet() {
    printf("Hello, User!");
}
```

---

5. Modifiers for Data Types

Modifiers **change the size or range** of the basic data types.

Modifier	Applied To	Description
`short`	`int`	Uses less memory (2 bytes)
`long`	`int, double`	Increases range (8 bytes for `long int`)
`signed`	`int, char`	Allows negative and positive values
`unsigned`	`int, char`	Only allows positive values

Example of Modifiers

```c
short int x = 10;     // Uses less memory
long int y = 100000; // Allows larger numbers
unsigned int z = 255; // Only positive numbers
```

---

6. Type Conversion (Implicit & Explicit)

C automatically converts data types in some cases (**Implicit conversion**).
Forcing conversion is called **Explicit conversion (Type Casting)**.

Example of Implicit Conversion

```c
int num = 10;
float result = num + 2.5; // 'num' (int) is automatically converted to float
```

Example of Explicit Conversion

```c
float pi = 3.14;
int intPi = (int) pi; // Casting float to int
```

Key Takeaways

- ✔ **Basic data types**: `int`, `char`, `float`, `double`
- ✔ **Derived types**: Arrays, Pointers, Structures
- ✔ **User-defined types**: `typedef`, `enum`
- ✔ **Modifiers**: `short`, `long`, `signed`, `unsigned`
- ✔ **Type conversion**: Implicit (automatic) & Explicit (type casting)

Would you like more examples or explanations on any of these topics?