1. **Left Shift Operator (`<<`)**
Definition:
The left shift operator (`<<`) shifts the bits of a variable to the left by a specified number of positions. For each shift to the left, the leftmost bits are discarded, and the rightmost bits are filled with zeros. It is equivalent to multiplying the number by `2^n`, where `n` is the number of positions shifted.
Syntax:
variable << number_of_positions;
Example:
int a = 5;   // In binary: 0000 0101
int result = a << 2; // Shifts bits of 'a' 2 positions to the left
// Result in binary: 0001 0100, which is 20 in decimal
Explanation:
- Initial value of `a` in binary: `0000 0101` (5 in decimal).
- After shifting 2 positions to the left: `0001 0100` (20 in decimal).

Numerical Questions:
1. What is the result of `7 << 3`?
   - **Explanation:** `7` in binary is `0000 0111`. Shifting 3 positions left: `0011 1000`, which is `56` in decimal.
   - **Answer:** `56`

2. Calculate `12 << 1`.
   - **Explanation:** `12` in binary is `0000 1100`. Shifting 1 position left: `0001 1000`, which is `24` in decimal.
   - **Answer:** `24`

2. **Right Shift Operator (`>>`)**
Definition:
The right shift operator (`>>`) shifts the bits of a variable to the right by a specified number of positions. For each shift to the right, the rightmost bits are discarded, and the leftmost bits are filled with the sign bit (for signed integers) or zeros (for unsigned integers). It is equivalent to dividing the number by `2^n`, where `n` is the number of positions shifted.
Syntax:
variable >> number_of_positions;
Example:
int b = 20;   // In binary: 0001 0100
int result = b >> 2; // Shifts bits of 'b' 2 positions to the right
// Result in binary: 0000 0101, which is 5 in decimal
Explanation:
- Initial value of `b` in binary: `0001 0100` (20 in decimal).
- After shifting 2 positions to the right: `0000 0101` (5 in decimal).

Numerical Questions:
1. What is the result of `40 >> 3`?

- **Explanation:** `40` in binary is `0010 1000`. Shifting 3 positions right: `0000 0101`, which is `5` in decimal.
- **Answer:** `5`

2. Calculate `16 >> 1`.
- **Explanation:** `16` in binary is `0001 0000`. Shifting 1 position right: `0000 1000`, which is `8` in decimal.
- **Answer:** `8`

Type Casting in C Language

Definition:
Type casting in C is the process of converting a variable from one data type to another. This can be done explicitly by the programmer or implicitly by the compiler. It is often used when you need to perform operations that require a specific data type or to avoid errors due to type mismatches.

Types of Type Casting:

1. **Implicit Casting (Automatic Casting):**
- **Definition:** The compiler automatically converts one data type to another when it is necessary, such as during arithmetic operations.
- **Example:** If you add an `int` to a `float`, the `int` is automatically converted to `float`.
```c
int a = 5;
float b = 4.5;
float result = a + b;  // `a` is implicitly cast to float
```

2. **Explicit Casting (Type Casting):**
- **Definition:** The programmer explicitly converts a variable from one type to another using a cast operator.
- **Syntax:** `(type) expression`
- **Example:**
```
int a = 10;
float b = (float)a / 3;  // `a` is explicitly cast to float
```

Numerical questions

Implicit Type Casting

1. **Question 1:**
```c
int a = 10;
float b = 3.5;
float result = a + b;
```
Answer: `result` will be `13.5` because `a` (int) is implicitly cast to `float`, and then the addition is performed.

2. **Question 2:**
```
int x = 7;
```

double y = x / 2;
Answer: `y` will be `3.0` because `x / 2` performs integer division, resulting in `3`, which is then implicitly cast to `double`.

3. **Question 3:**
char c = 'A';
int num = c + 10;
Answer: `num` will be `75`. The ASCII value of `'A'` is `65`, so `65 + 10 = 75`.

4. **Question 4:**
double d = 9.99;
int i = d / 2;
Answer: `i` will be `4` because `d / 2` results in `4.995`, which is implicitly cast to `int` as `4` (fractional part is discarded).

5. **Question 5:**
float f = 4.5;
int result = f * 2;
Answer: `result` will be `9` because `f * 2` equals `9.0`, which is implicitly cast to `int` as `9` (fractional part is discarded).

Explicit Type Casting
1. **Question 1:**
float f = 5.7;
int i = (int)f;
Answer: `i` will be `5` because `(int)f` explicitly casts the float value `5.7` to `int`, discarding the fractional part.

2. **Question 2:**
int x = 9;
float y = (float)x / 4;
Answer: `y` will be `2.25`. `(float)x` converts `x` to `float`, so `9.0 / 4 = 2.25`.

3. **Question 3:**
double d = 8.5;
int i = (int)(d / 2);
Answer: `i` will be `4` because `d / 2` equals `4.25`, and `(int)` casts it to `4` (fractional part is discarded).

4. **Question 4:**
char c = 'Z';
int num = (int)c;
Answer: `num` will be `90`, as the ASCII value of `'Z'` is `90`.

5. **Question 5:**
int a = 15;
float f = (float)a / 4;
Answer: `f` will be `3.75` because `(float)a` converts `a` to `float`, so `15.0 / 4 = 3.75`.