Question 1

```
int fun(int n) {
    if(n == 0)
        return 0;
    return n + fun(n - 1);
}
int main() {
    printf("%d", fun(4));
}
```

Output: 10

Question 2

```
int fun(int n) {
    if(n <= 1)
        return 1;
    return n * fun(n - 1);
}
int main() {
    printf("%d", fun(4));
}
```

Output: 24 // factorial

Question 3

```
int fun(int n) {
    if(n < 0)
        return 0;
    return 1 + fun(n - 2);
}
int main() {
    printf("%d", fun(5));
}
```

Output: 3

Question 4

```
int fun(int n) {
    if(n == 1)
```

```
        return 1;
    return fun(n / 2) + n;
}
int main() {
    printf("%d", fun(5));
}
```

Output: 8

Question 5

```
int fun(int n) {
    if(n == 0)
        return 1;
    return fun(n - 1) + fun(n - 1);
}
int main() {
    printf("%d", fun(3));
}
```

Output: 8 // 2^3

Question 6

```
int fun(int n) {
    if(n <= 0)
        return 0;
    return fun(n - 1) + 2;
}
int main() {
    printf("%d", fun(4));
}
```

Output: 8

Question 7

```
int fun(int n) {
    if(n == 0)
        return 0;
    return (n % 10) + fun(n / 10);
}
int main() {
    printf("%d", fun(123));
}
```

Output: 6 // sum of digits

Question 8

```
int fun(int n) {
    if(n == 1)
        return 2;
    return fun(n - 1) * 2;
}
int main() {
    printf("%d", fun(4));
}
```

Output: 16

Question 9

```
int fun(int n) {
    if(n <= 1)
        return n;
    return fun(n - 1) + fun(n - 2);
}
int main() {
    printf("%d", fun(6));
}
```

Output: 8 // Fibonacci

Question 10

```
int fun(int n) {
    if(n == 0)
        return 1;
    if(n % 2 == 0)
        return fun(n - 1);
    return fun(n - 1) + n;
}
int main() {
    printf("%d", fun(5));
}
```

Output: 10

Question 11

```
int fun(int n) {
    if(n == 1)
        return 1;
    return n + fun(n / 2);
}
int main() {
    printf("%d", fun(7));
```

```
}
```

Output: 11

Question 12

```
int fun(int n) {
    if(n == 0)
        return 0;
    return fun(n - 1) * 10 + 1;
}
int main() {
    printf("%d", fun(3));
}
```

Output: 111

Question 13

```
int fun(int n) {
    if(n <= 0)
        return 0;
    return fun(n - 2) + n;
}
int main() {
    printf("%d", fun(6));
}
```

Output: 12

Question 14

```
int fun(int n) {
    if(n == 0)
        return 0;
    return fun(n - 1) + n * n;
}
int main() {
    printf("%d", fun(3));
}
```

Output: 14 // $1^2 + 2^2 + 3^2$

Question 15

```
int fun(int n) {
    if(n == 1)
```

```
        return 3;
    return fun(n - 1) + 3;
}
int main() {
    printf("%d", fun(4));
}
```

Output: 12

Question 16

```
int fun(int n) {
    if(n <= 1)
        return 1;
    return n * fun(n - 2);
}
int main() {
    printf("%d", fun(5));
}
```

Output: 15 // 5 * 3 * 1

Question 17

```
int fun(int n) {
    if(n < 10)
        return n;
    return fun(n / 10) + n % 10;
}
int main() {
    printf("%d", fun(256));
}
```

Output: 13

Question 18

```
int fun(int n) {
    if(n == 0)
        return 0;
    return (n & 1) + fun(n >> 1);
}
int main() {
    printf("%d", fun(13));
}
```

Output: 3 // count of set bits in 13 (1101)

Question 19

```
int fun(int n) {
    if(n <= 1)
        return 1;
    return fun(n - 1) - fun(n - 2);
}
int main() {
    printf("%d", fun(5));
}
```

Output: `0`

Question 20

```
int fun(int n) {
    if(n == 0)
        return 1;
    return fun(n - 1) + fun(n - 1) + fun(n - 1);
}
int main() {
    printf("%d", fun(3));
}
```

Output: 27 // 3^3

Question:Difference between break and continue with Example

`break` Statement — Terminate the loop

Meaning:

`break` is used to **immediately stop the execution of a loop** (`for`, `while`, `do-while`) or to exit from a `switch` statement.

Once `break` is executed:

- The loop ends.
- Control jumps to the **first statement after the loop**.

When to use:

- When the required result is found.
- When a condition is met and further looping is unnecessary.

- To avoid extra computations.

Example:

```
#include <stdio.h>

int main() {

    for(int i = 1; i <= 5; i++) {

        if(i == 3)

            break;

        printf("%d ", i);

    }

    return 0;

}
```

Output:

1 2

Explanation:

When i becomes 3, break stops the loop completely.

Important points about `break`:

- Exits only the nearest enclosing loop.
- Commonly used in searching problems.
- In `switch`, it prevents fall-through.

`continue` Statement — Skip current iteration

Meaning:

`continue` is used to skip the remaining statements of the current iteration and move to the next iteration of the loop.

When `continue` is executed:

- The loop does not end.
- Control goes back to the next iteration.

When to use:

- When some values should be ignored.
- When you want to process only certain cases.

Example :

```
#include <stdio.h>

int main() {

    for(int i = 1; i <= 5; i++) {

        if(i == 3)

            continue;

        printf("%d ", i);

    }

    return 0;

}
```

Output:

```
1 2 4 5
```

Explanation:
When `i` is 3, that iteration is skipped, but the loop continues.

Important points about `continue`:

- Skips only the current iteration, not the entire loop.

- Useful for filtering values.
- Works only with loops, not with `switch`.