21-12-25

MCQ’s on Recursion in C

1. Output of simple recursion

Solve this: Find the output

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

A) 3
B) 6
C) 9
D) 0

2. Recursive function call count

Solve this: How many times is fun() called?

```
void fun(int n){
   if(n == 0)
      return;
   fun(n-1);
}
fun(5);
```

A) 4
B) 5
C) 6
D) Infinite

3. Output with return multiplication

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

A) 10
B) 24
C) 16
D) 6

4. Missing base case

Solve this: What happens?

```
int fun(int n){
   return fun(n-1);
}
fun(5);
```

A) Returns 0
B) Returns garbage value
C) Infinite recursion
D) Compile-time error

5. Output with post-recursion print

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

```
fun(3);
```

A) 3 2 1
B) 1 2 3
C) 3 1 2
D) No output

6. Output with pre-recursion print

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

A) 1 2 3
B) 3 2 1
C) 2 3 1
D) Error

7. Recursion with condition

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

A) 3
B) 4
C) 5
D) 6

8. Output order

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

A) 1 2 1
B) 2 1 2
C) 1 1 2
D) 2 2 1

9. Output with double recursion

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

A) 1 2
B) 1 1 2
C) 2 1 1
D) 1 2 2

10. Missing return value

```
int fun(int n){
```

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

A) 1
B) 3
C) Garbage value
D) Compile error

11. Output with decrement

```
int fun(int n){
    if(n == 0)
        return 0;
    return fun(n/2) + 1;
}
printf("%d", fun(8));
```

A) 3
B) 4
C) 8
D) 1

12. Output prediction

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

A) 5 4 3 2 1
B) 5 3 1
C) 1 3 5
D) 5 2

13. How many times is fun() called?

```
int fun(int n){
   if(n == 0)
      return 0;
   fun(n-1);
   fun(n-1);
}
fun(3);
```

A) 7
B) 8
C) 6
D) 4

14. Output with static variable

```
int fun(){
   static int x = 0;
   x++;
   if(x <= 3){
      fun();
   }
   return x;
}
printf("%d", fun());
```

A) 1
B) 2

C) 3
D) 4

15. Output with tail recursion

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

A) 3
B) 6
C) 9
D) 0

16. Infinite recursion check

```
void fun(int n){
   if(n == 0)
      fun(n);
}
fun(1);
```

A) Runs once
B) Compile error
C) Infinite recursion
D) No output

17. Output order

```
void fun(int n){
   if(n == 0)
      return;
   fun(n-1);
```

```
    printf("%d ", n);
    fun(n-1);
}
fun(2);
```

A) 1 2 1
B) 2 1 2
C) 1 1 2
D) 2 2 1

18. Static variable trap

```
int fun(){
    static int x = 0;
    x++;
    if(x <= 2)
        fun();
    return x;
}
printf("%d", fun());
```

A) 1
B) 2
C) 3
D) Infinite

19. Output with return value

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

A) 1
B) 5
C) 0
D) Garbage value

20. Find the output

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

A) 2 1 1 2
B) 1 2 2 1
C) 2 1 2 1
D) 1 2 1 2

Explain difference between break and continue

🔹 **break — Exit the loop completely**

📌 **Concept**

- break **terminates the entire loop immediately**.
- Control jumps **outside the loop**, to the next statement after it.
- Used when **no further iterations are needed**.

🧠 **How it works**

- Stops loop execution **permanently**
- Loop condition is **not checked again**

🔍 **Where it is used**

- Inside for, while, do-while, and switch

- Common in **searching**, **menu-driven programs**, **early exit**

🧪 **Example**

```
for(int i = 1; i <= 5; i++){
        if(i == 3)
            break;
        printf("%d ", i);
}
```

Output:

1 2

➞ When i == 3, loop ends completely.

🔹 **continue — *Skip current iteration only***

📌 **Concept**

- continue **skips the remaining statements of the current iteration**.
- Control jumps to the **next iteration** of the loop.
- Loop **does not terminate**.

🧠 **How it works**

- Current iteration stops
- Loop condition is checked again

🔍 **Where it is used**

- Inside loops only (for, while, do-while)
- Used to **ignore specific values** or **filter data**

🧪 **Example**

```
for(int i = 1; i <= 5; i++){
    if(i == 3)
        continue;
    printf("%d ", i);
}
```

Output:

1 2 4 5

➞ When i == 3, printing is skipped, loop continues.