C Aptitude Questions

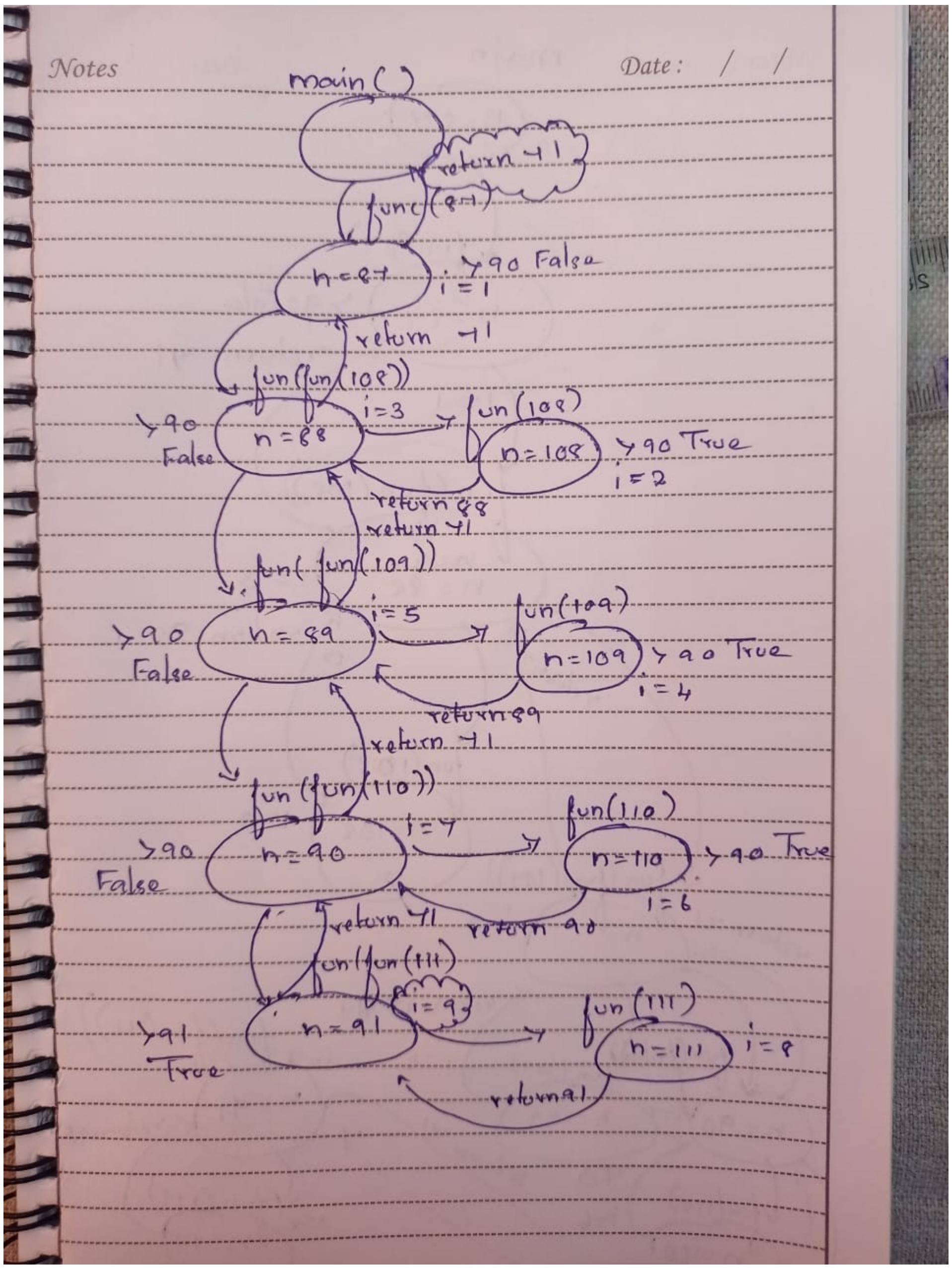
1.

```c
#include<stdio.h>

int i = 0;
int fun(int n){
    i++;
    if(n > 90)
        return n-20;
    return fun(fun(n + 21));
}

void main(){
    printf("%d \n", fun(87));
    printf("%d\n", i);
}
```

Trace the recursive calls to find the solution
here i am adding my version of tracing the recursive calls
The output is
71
9



2.

```
void main(){
    int i = 4, j = 8;
    i = i|j & j|i + i|j & j|i - i^j;
    j = i|i & j|j + j|j & i|i - j^j;
    printf("%d %d %d\n", i|j & j|i, i|j &j|i, i^j);
}
```

The question is related to operator precidence....
See the following operator precidence table

Category	**Operator**	**Associativity**
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= ;=	Right to left
Comma	,	Left to right

The operations to be performed is of the order
* Addition/Subraction
* Bitwise AND
* Bitwise XOR
* Bitwise OR

so the expression

i = i|j & j|i + i|j & j|i – i^j; -> i = 4|8 & 8|4 + 4|8 & 8|4 – 4^8; -> 4|8 & 8| 8 |8 & 8| 0 ^8; -> 4 | 8 | 8 | 8 | 0 ^8

-> 4 | 8 | 8 | 8 | 8 -> 4 | 8 -> 12

so the updated value of i would be 12

next expression

j = i|i & j|j + j|j & i|i – j^j; -> 12|12 & 8|8 + 8|8 & 12|12 – 8^8 -> 12|12 & 8| 16 |8 & 12| 4 ^8 ->

12| 8 | 16 | 8 | 4 ^8 -> 12| 8 | 16 | 8 | 12 -> 12 | 8 | 16 -> 28

So the updated value of j would be 28

lastly the expression in the print statements

i|j & j|i -> 12|28 & 28|12 -> 12| 28 | 12 -> 12|28 -> 28

i^j -> 12 ^ 28 -> 16

So the output would be
28 28 16

3.

```
#include<stdio.h>

int main(){
    char *str = "Hello World";
    int i;
    int len = strlen(str);
    for(i = 0; i<= len; i++){
        printf("%c", str[len - i]);
    }
    return 0;
}
```

Output:

```
\0dlroW olleH
         (or)
dlroW olleH
```

Explanation:
The program is to print the string in reverse order....

In c string/arrays are usually terminated with /0 (Empty spaces in array are usually filled with /0)

So the output is dependent on terminal and both outputs are valid

4.

```
#include<stdio.h>

struct module1{

    unsigned int x : 5;
    unsigned int y: 8;

}m1;

struct module2{

    unsigned int x:5;
    unsigned int : 0;
    unsigned int y : 8;

}m2;

int main(){
    printf("%d %d\n", m1.x++ - ++m2.x, m1.y + m2.y++);
    return 0;
}
```

Output

```
-1 0
```

Explanation

The wierd statements like `unsigned int x : 5;` does not have any significance to the output you can learn about these statements here -> C bit fields

when a structure variable is created, the members inside the structure is initialized

if
int -> 0
float -> 0
char -> \0;
m1.x, m1.y, m2.x, m2.y = 0
so on valuating

```
        m1.x++ - ++m2.x -> 0 – 1 -> -1
        m1.y + m2.y++ -> 0 + 0 -> 0
```

5.

```
#include<stdio.h>
int main() {
    int i =0;
    for(i = 0; i< 20; i++){
        switch(i){

            case 0:
            i += 5;

            case 1:
            i += 2;

            case 5:
            i += 5;

            default:
            i += 4;
            break;

        }
        printf("%d ", i);
    }
    return 0;
}
```

Output :

```
16 21
```

Explanation:

iteration 1
i = 0
case 0 True
 i +=4 => 5
as there is no break statement the subsequent lines will be executed

i += 2 => 7
i += 5 => 12
i += 4 => 16 // prints 16

iteration 2
i = 17
none of the cases matches so the default case is executed...
i +=4 => 21 // prints 21

Note : The break statements inside a switch does not terminate the loops rather terminates the switch statement

6.

```
#include <stdio.h>
#define square(x) x*x

int main(){
    int i;
    i = 64 / square(4);
    printf("%d \n", i);
}
```

Output
```
64
```

Explanation:
#define is used to define preprocessor directives ie., when used the defined name the name is replaced with the constant or expression

here we defined square(x) as x*x
so in i = 64 / square(x) -> 64 / x * x -> 64 / 4 * 4 -> 16 * 4 = 64.

7. Considering int occupies 2 bytes of space predict the output of the following

```
#include<stdio.h>

int main(void) {

    int i = 3;
    int j;
    j = sizeof(++i + ++i);
    printf("i = %d j = %d\n", i, j);
    return 0;

}
```

Output
```
i = 3 j = 2
```

Explanation:
sizeof is an operator and not a function and it only evaluates an expression if truely needed so in this case it does not evaluate the expression
for more details about sizeof Why size of does not increment x

8.

```
#include<stdio.h>
int main(){
    char s[] = "zoho";
    int i;

    for(i = 0; s[i]; i++)
        printf("\n%c %c %c %c", s[i], *(s + i), *(i + s), i[s]);
}
```

Output:

```
z z z z
o o o o
h h h h
o o o o
```

Explanation:

Compilers use pointer arithmetic internally to access array elements.

ie., the expression a[i] is evaluated as *(a + i) so the expressions

s[i], *(s + i), *(i + s), i[s] are all same

9.

```
#include<stdio.h>

int main() {
    void fun(char*);
    char a[100];

    a[0] = 'Z'; a[1] = 'O';
    a[2] = 'H'; a[3] = 'O';

    fun(&a[0]);
    return 0;
}

void fun(char *a){
    a++;
    printf("%c", *a);
    a++;
    printf("%c", *a);
}
```

Output:

OH
Explanation:

the line void fun(char*); -> is called function prototype declaration
it tells the compiler that there exists a function named `fun` with an parameter of type character pointer. This line dosen't have any specific meaning to the output

then the address of 0th index is passed as an argument to the functioin fun
the function then computes a++ so now a points to the 1st index of the string zoho and prints o again a++ and now a points to the 2nd index of the string. Now in the print statement h is printed

10.

```
#include<stdio.h>
#include<string.h>

int main(){
    char str1[] = "zohocorp.com";
    char str2[20] = "";

    strncpy(str2, str1, 8);
    printf("%s", str2);
    return 0;
}
```

Output:
zohocorp

Explanation:
strncpy -> The C library function char *strncpy(char *dest, const char *src, size_t n) copies up to n characters from the string pointed to, by src to dest.
Here n -> 8 so first 8 characters from str1 is copied to str2 and then it is printed