

GATE 2015 — C PROGRAMMING

Question 34 • NAT • Functions, Pass-by-Value & Pointers

Final Answer

-5

Question

The output of the following C program is _____.

```
void f1(int a, int b) {
    int c;
    c = a;  a = b;  b = c;
}

void f2(int *a, int *b) {
    int c;
    c = *a;  *a = *b;  *b = c;
}

int main() {
    int a = 4, b = 5, c = 6;
    f1(a, b);
    f2(&b, &c);
    printf("%d", c-a-b);
}
```

1. Start with main()

```
int a = 4, b = 5, c = 6;
f1(a, b);
f2(&b, &c);
printf("%d", c-a-b);
```

The execution starts from main(). Three local variables are created: a = 4, b = 5 and c = 6.

Initial values in main()

a = 4 b = 5 c = 6

2. Understand f1(a, b)

```
void f1(int a, int b) {
```

```
int c;
c = a;
a = b;
b = c;
}
```

The call is `f1(a, b)`. The values 4 and 5 are passed to `f1()`. For ordinary C parameters such as these, the function receives copies of the values.

Think of `f1()` as having its own local copies

Inside `f1()`: `a = 4`, `b = 5`. The statements swap these local copies, making `a = 5` and `b = 4` inside `f1()`.

The important point is that `main()`'s `a` and `b` are not changed by this call.

After `f1(a, b)`

`main()`: `a = 4`, `b = 5`, `c = 6`

3. Can we ignore `f1(a, b)`?

The function `f1()` is actually executed, so it should not be described as being skipped or ignored.

However, for calculating the final output, its effect on `main()`'s variables is zero. Therefore, after tracing it once, we can ignore its effect on the final calculation.

Exam shortcut

`f1(a,b)` → changes only local copies → `main()` remains unchanged.

4. Understand `f2(&b, &c)`

```
f2 (&b, &c) ;
```

Here `&b` means the address of `main()`'s `b`, and `&c` means the address of `main()`'s `c`. Therefore, `f2()` receives pointers to the original variables.

Pointer house analogy

Think of a variable as a HOUSE. The variable's address is the HOUSE ADDRESS. A pointer stores that address. `*pointer` means: go to that address and access the value stored there.

```
void f2(int *a, int *b) {
    int c;
    c = *a;
    *a = *b;
    *b = c;
}
```

5. Trace f2() line by line

Because f2(&b, &c) was called, inside f2():

```
a → address of main()'s b
b → address of main()'s c
*a → value of main()'s b
*b → value of main()'s c
```

Initially, main() has b = 5 and c = 6.

First statement: c = *a; → local c becomes 5.

Second statement: *a = *b; → main()'s b becomes 6.

Third statement: *b = c; → main()'s c becomes 5.

Effect of f2()

f2() swaps the values of main()'s b and c.

Values after f2()

main(): a = 4, b = 6, c = 5

6. Evaluate printf()

```
printf("%d", c-a-b);
```

Use the final values from main(): c = 5, a = 4 and b = 6.

```
c - a - b
= 5 - 4 - 6
= 1 - 6
= -5
```

Output

-5

7. Function Basics — Quick Revision

Calling function: The function that makes the call. Here, main() is the calling function.

Called function: The function that is called. Here, f1() or f2() is the called function.

Function definition: The function header plus the statements inside its braces.

When a function finishes, control returns to the calling function. A void function may finish by reaching its closing } or by executing return;

A non-void function such as int, float or char normally returns a value using return value;.

Local automatic variables have a lifetime that ends when their function execution ends.

8. Pointer Quick Rule

☆ Easy memory trick

Variable = 🏠 HOUSE | &variable = 📍 HOUSE ADDRESS | pointer = 📄 stores the address |
 *pointer = 📄 access the value at that address

9. Complete Solution in One View

```
main()
↓
a=4, b=5, c=6
↓
f1(a,b)
→ only copies are changed
→ main remains: 4,5,6
↓
f2(&b,&c)
→ b and c are swapped
→ main becomes: a=4, b=6, c=5
↓
c=a-b
= 5-4-6
= -5
```

🏆 GATE Takeaway

The question combines two ideas: passing ordinary variables to a function gives the function values to work with, while passing addresses with & allows pointer parameters to modify the original variables.

GATE Aspirants
Keep Going...

“ Consistent Practice Today Creates a Brighter Tomorrow. ”

Clear Concepts | Solve More Problems | Learn Step by Step | Improve Regularly | Achieve Your Goal

EngineersTutor.com
Your Learning Partner for a Brighter Future

Like | Share | Subscribe

DISCIPLINE PRACTICE PATIENCE SUCCESS