**Variable Scope and Storage Specifier Assignments**

1. Refer the code snippet below and answer the queries

int val;

extern void display();

static int function()

{

val++;

int x = 10;

int i = 0;

static int j = 20;

for (; i < 3; i++)

{

int x = 20;

printf(“\n %d”, x+i);

x+=3;

j++;

display();

}

return val;

}

int main(int argc, char \*argv[])

{

val= 0;

function();

return 0;

}

**a. What is the change required if val declaration line below is to be moved to another file?**

If val is to be moved to another file (i.e., another source file), you need to declare it as extern in the file where function() is defined so that the compiler knows that the variable val is declared and defined elsewhere.

In the file where function() is located, you would change the declaration of val to:

c

Copy code

extern int val;

This tells the compiler that val is defined in some other file (likely the one containing the main function).

**b. What is the value of x after the for loop execution?**

In the code, there are two variables named x:

1. The x declared at the beginning of the function() with the statement int x = 10;.
2. A new x declared inside the for loop with int x = 20;, which **shadows** the outer x.

The inner x inside the loop gets re-initialized to 20 on each iteration of the loop and is modified within the loop (x += 3;), but the changes to this inner x do not affect the outer x. Thus, the outer x (with initial value 10) remains unchanged after the loop.

So, after the loop execution, the value of the outer x is **10**.

**c. What does the keyword static in the following lines mean?**

The static keyword in C has different meanings depending on where it is used:

1. **static int function()**:  
   This means the function is limited to the scope of the current file. It cannot be called from other files because its linkage is internal to the file where it is defined. This helps to avoid name conflicts and makes the function private to the file.
2. **static int j = 20;**:  
   The static keyword here means that the variable j retains its value between function calls. Unlike local variables which are destroyed and reinitialized each time a function is called, static variables are initialized only once and retain their value throughout the program's execution. In this case, j will be incremented on each iteration of the loop and retain its value across successive calls to function().

**d. What is the value of j after the for loop execution?**

j is a static variable initialized to 20. It is incremented each time through the loop. The loop runs 3 times (for i = 0, 1, 2), so j is incremented 3 times.

* Initial value of j = 20
* After 1st iteration: j = 21
* After 2nd iteration: j = 22
* After 3rd iteration: j = 23

Therefore, after the for loop execution, the value of j is **23**.

**e. Identify the variables which would be in the stack of function()**

In the stack, there are **local variables** and **static variables**. For the function function():

1. **Local variables (on the stack)**:
   * int x = 10; (outer x)
   * int i = 0;
   * int x = 20; (inner x in the for loop, which is re-initialized on each iteration)
   * int j = 20; (This is a static variable, so it will **not** be on the stack but stored in a separate memory section dedicated to static data.)
2. **Static variables**:
   * static int j = 20; (This does not go on the stack but is stored in a static data section and retains its value across calls to the function.)

So, the stack will contain the following variables:

* x (local to the function)
* i (loop counter)
* The inner x in the loop, which is shadowed and re-initialized every time
  + Static variables, like j, are not part of the stack but reside in static memory.

**f. What does extern in the following line mean?**

c

Copy code

extern void display();

The extern keyword here indicates that the display() function is declared, but its definition is **not in this file**. Instead, it is assumed to be defined elsewhere, possibly in another file that is linked during the program's compilation process. The compiler knows that the function display() exists, but it will resolve its actual location at the linking stage.

In summary:

* extern tells the compiler to look for the actual definition of display() in another file.