#include<iostream>

using namespace std;

void update(int \*\*p2) {

// p2 = p2 + 1;

// kuch change hoga - NO

// This line attempts to increment the pointer `p2` (which is itself a pointer to another pointer). However, this change is local to the function and does not affect the pointer `p2` in `main`.

// \*p2 = \*p2 + 1;

// kuch change hoga - YES

// This line increments the pointer `\*p2` (the pointer `p` in `main`), effectively changing where `p` points. It would make `p` point to the next integer in memory.

\*\*p2 = \*\*p2 + 1;

// kuch change hoga - YES

// This line increments the value at the address pointed to by `p` (which is stored in `p2`), thus modifying the integer `i` that `p` points to.

}

void update(int \*p) {

\*p = (\*p) \* 2;

// This function doubles the value of the integer pointed to by `p`.

}

void increment(int \*\*p) {

++(\*\*p);

// This function increments the value of the integer pointed to by the pointer `\*p` (which is itself a pointer).

}

int main() {

/\*

int i = 5;

int\* p = &i;

int\*\* p2 = &p;

// Demonstrating pointer and double pointer operations

cout<< endl << endl <<" Sab sahi chal rha h " << endl << endl ;

// Outputs the value of `i` in three different ways

cout << i << endl; // 5

cout << \*p << endl; // 5

cout << \*\*p2 << endl; // 5

// Outputs the memory address of `i`, `p`, and `p2`

cout << &i << endl; // Address of `i`

cout << p << endl; // Address of `i`

cout << \*p2 << endl; // Address of `i`

cout << &p << endl; // Address of `p`

cout << p2 << endl; // Address of `p`

// Update through a double pointer

cout<< "before " << i << endl; // 5

cout<< "before " << p << endl; // Address of `i`

cout<< "before " << p2 << endl; // Address of `p`

update(p2); // Increment the value of `i`

cout<< "after " << i << endl; // 6 (after increment)

cout<< "after " << p << endl; // Address of `i`

cout<< "after " << p2 << endl; // Address of `p`

\*/

int num = 110;

int \*ptr = &num;

increment(&ptr); // This increments the value of `num` by 1

cout << num << endl; // Outputs: 111

return 0;

}

**Detailed Breakdown:**

1. **Understanding the update Function**:
   * **Pointer to Pointer (int \*\*p2)**:
     + p2 = p2 + 1; – This operation attempts to move the double pointer p2 to the next memory address, but since this change is local to the function, it doesn't affect the original pointer in main.
     + \*p2 = \*p2 + 1; – This operation would move the pointer p to the next memory address (assuming p2 points to p), effectively changing where p points. This change affects the pointer p in main.
     + \*\*p2 = \*\*p2 + 1; – This line modifies the value of i by incrementing it, as \*\*p2 dereferences p2 to get the pointer p, and then dereferences p to get the value of i.
2. **Second update Function**:
   * **Pointer Manipulation (int \*p)**:
     + The function update(int \*p) directly modifies the value pointed to by p. In the example, if p pointed to i, it would double the value of i.
3. **Increment Function**:
   * The increment(int \*\*p) function increments the value of i by 1, demonstrating how changes through a double pointer affect the original variable.
4. **Main Function**:
   * **Double Pointers**:
     + The code in the main function (currently commented out) sets up a double pointer p2 to point to p, which in turn points to i. This demonstrates how double pointers can be used to indirectly modify the value of a variable (i).
   * **Incrementing num**:
     + The line increment(&ptr); increments the value of num by 1, demonstrating how a pointer to a pointer can be used to modify the value of the original variable indirectly.

**Key Concepts:**

* **Double Pointers**:
  + Double pointers (int \*\*p2) are pointers that hold the address of another pointer. They are useful for modifying not just the value of a variable but also the pointer itself (e.g., changing where a pointer points).
* **Pointer Arithmetic**:
  + You can perform arithmetic operations on pointers, such as incrementing a pointer to move to the next memory address. However, these changes are typically local to the function unless the pointer is returned or modified indirectly through a double pointer.
* **Dereferencing**:
  + Dereferencing a double pointer (\*\*p2) accesses the value stored at the memory address the original pointer (p) points to.