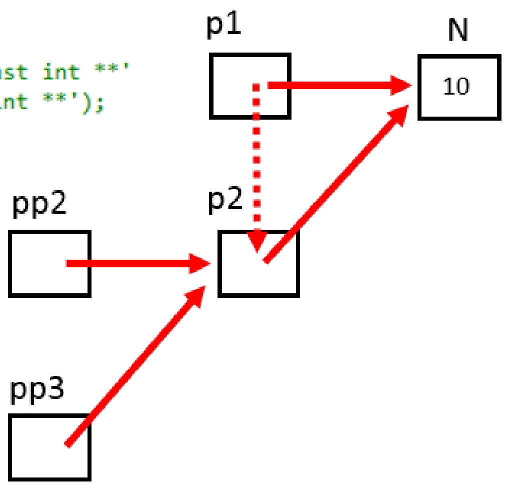
QUESTION:
Why is it not possible to make a call to function **func2()** below, using an **int **** as argument of the call, when the parameter of the function is **const int **** ?
ANSWER:
Because, if that were possible, it would be possible for **func2()** to modify the value of the integer pointed indirectly. So the compiler forces that the prototype must be **void func2(const int * const * x)**, that is **x** is **pointer to const pointer to const int**; in this way the "intermediate" pointer cannot be modified, avoiding that the pointed value is modified, inside **func2()**.
See the code below and the drawing in next page for an explanation

```
=============================================================================================
#include <iostream>
using namespace std;

void func1(const int* x)
{
  cout << *x << endl;
}

// In void func2(const int ** x)
// 'x' is a pointer to pointer to 'const int' (so, it shouldn't be possible to modify the indirectly pointed int value)
// but it would be, as illustrated in the example in annex, because of the existence of a "pointer in the middle"
// It is to avoid the modification, using the "pointer in the middle", that the compiler gives an error when compiling this function.
// To avoid the compiling error, the prototype of the function must be replaced with:
// void func2(const int * const * x) // declare x as 'pointer to const pointer to const int'
// so, the "pointer in the middle" cannot be modified
void func2(const int ** x) // 'x' is a pointer to pointer to 'const int'
{
  cout << **x << endl;
}

int main()
{
  // "UNCOMMENT" THE FOLLOWING BLOCKS OF CODE, ONE AT A TIME, AND TRY TO COMPILE

  //--------------------------------------------------------------------------------
  // BLOCK 1
  int x = 1;
  int* ptr1 = &x;
  int** ptr2 = &ptr1;
  func1(ptr1); // pointer to int can be converted to a const func parameter
  func2(ptr2); // pointer to pointer to int can't

  ////------------------------------------------------------------------------------
  //// BLOCK 2
  ////// CODE FROM SLIDE - VERSION 1

  //const int N = 10;
  //const int *p1 = &N;
  //int *p2;
  //int **pp2 = &p2;
  //// THE NEXT STATEMENT WOULD BE EQUIVALENT TO PASS A 'int **' TO A 'const int **'
  //// It doesn't compile (ERROR: cannot convert from 'int **' to 'const int **');
  //const int ** pp3 = pp2;
  //// BUT IF WOULD, THEN YOU COULD DO:
  //*pp3 = p1;  // equivalent to p2 = p1;
  //*p2 = 20;  // equivalent to N = 20 ...! but N is const!!!

  ////------------------------------------------------------------------------------
  // BLOCK 3
  //// CODE FROM SLIDE - VERSION 2

  //const int N = 10;
  //const int *p1 = &N;
  //int *p2;
  //int **pp2 = &p2;
  //const int * const * pp3 = pp2;
  //// WHAT HAPPENS IF YOU TRY TO USE INTERMEDIATE POINTER p2 TO ACCESS THE CONSTANT 'N':
  //*pp3 = p1;  // equivalent to p2 = p1; => ERROR: you cannot assign to a variable that is const
  //*p2 = 20;

  return 0;
}
```

THIS CODE GIVES A COMPILATION ERROR BUT ILLUSTRATES WHAT IT WOULD BE POSSIBLE TO DO IF IT WAS POSSIBLE TO ASSIGN AN int ** TO AN const int **!!!
FOLLOW THE CODE WITH THE DIAGRAM NEXT

```
const int N = 10;
const int *p1 = &N;
int *p2;
int **pp2 = &p2;
// THE NEXT STATEMENT WOULD BE EQUIVALENT TO PASS A 'int **' TO A 'const int **'
// It doesn't compile (ERROR: cannot convert from 'int **' to 'const int **');
const int **pp3 = pp2;
// BUT IF WOULD, THEN YOU COULD DO:
*pp3 = p1;  // equivalent to p2 = p1;
*p2 = 20;  // equivalent to N = 20 ...! but N is const!!!
```



THIS CODE ALSO GIVES A COMPILATION ERROR, BUT SHOWS THAT WHEN pp3 IS DECLARED AS A const int * const *
IT IS NO LONGER POSSIBLE TO MODIFY THE VALUE THAT IS POINTED INDIRECTLY USING THE INTERMEDIATE POINTER p2

```
const int N = 10;
const int *p1 = &N;
int *p2;
int **pp2 = &p2;
const int * const * pp3 = pp2;
// WHAT HAPPENS IF YOU TRY TO USE INTERMEDIATE POINTER p2 TO ACCESS THE CONSTANT 'N':
*pp3 = p1;  // equivalent to p2 = p1; => ERROR: you cannot assign to a variable that is const
*p2 = 20;
```