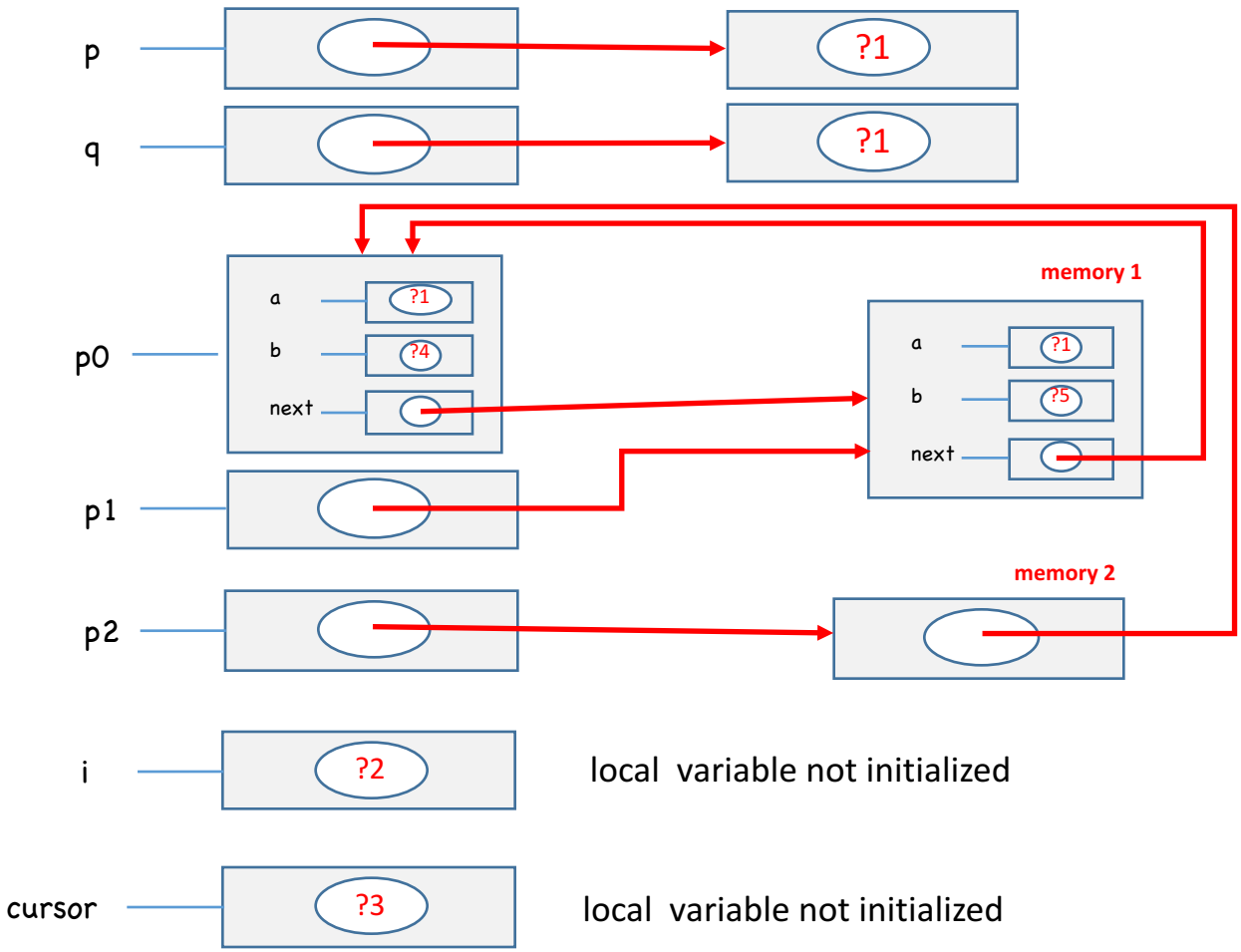
CSE 340 Fall 2019 Homework 4 Solution

```
/ CSE340 Fall 2019 HOMEWORK 4
//
// DUE DATE: Wednesday November 6 2019 by 11:59 pm on canvas/gradescope
//
// This code contains a number of questions.
// You should answer the questions and provide justification for your answers.
// All questions assume that the code is compiled according to ANSI C99
// standard. To compile it according to C99 standard execute the
// following from the command prompt (not visual studio!)
//
//        gcc -std=gnu99 CSE340S18_HW4.c
//

#include <stdio.h>
#include <stdlib.h>

struct T {
            int a;
            int *b;
            struct T *next;
} ;

struct T p0;
struct T *p1;
struct T **p2;
struct T **p3;

int *p;
int *q;

int main()
{
       int i;
       struct T* cursor;

       p1 = (struct T *) malloc(sizeof(struct T));
       p2 = (struct T **) malloc(sizeof(struct T *));

       p = (int *) malloc(sizeof(int));
       q = (int *) malloc(sizeof(int));

       *p = *q;

       p0.next = p1;
       (*p1).a = *p;
       (*p1).next = &p0;
       *p2 = (*p1).next;
```

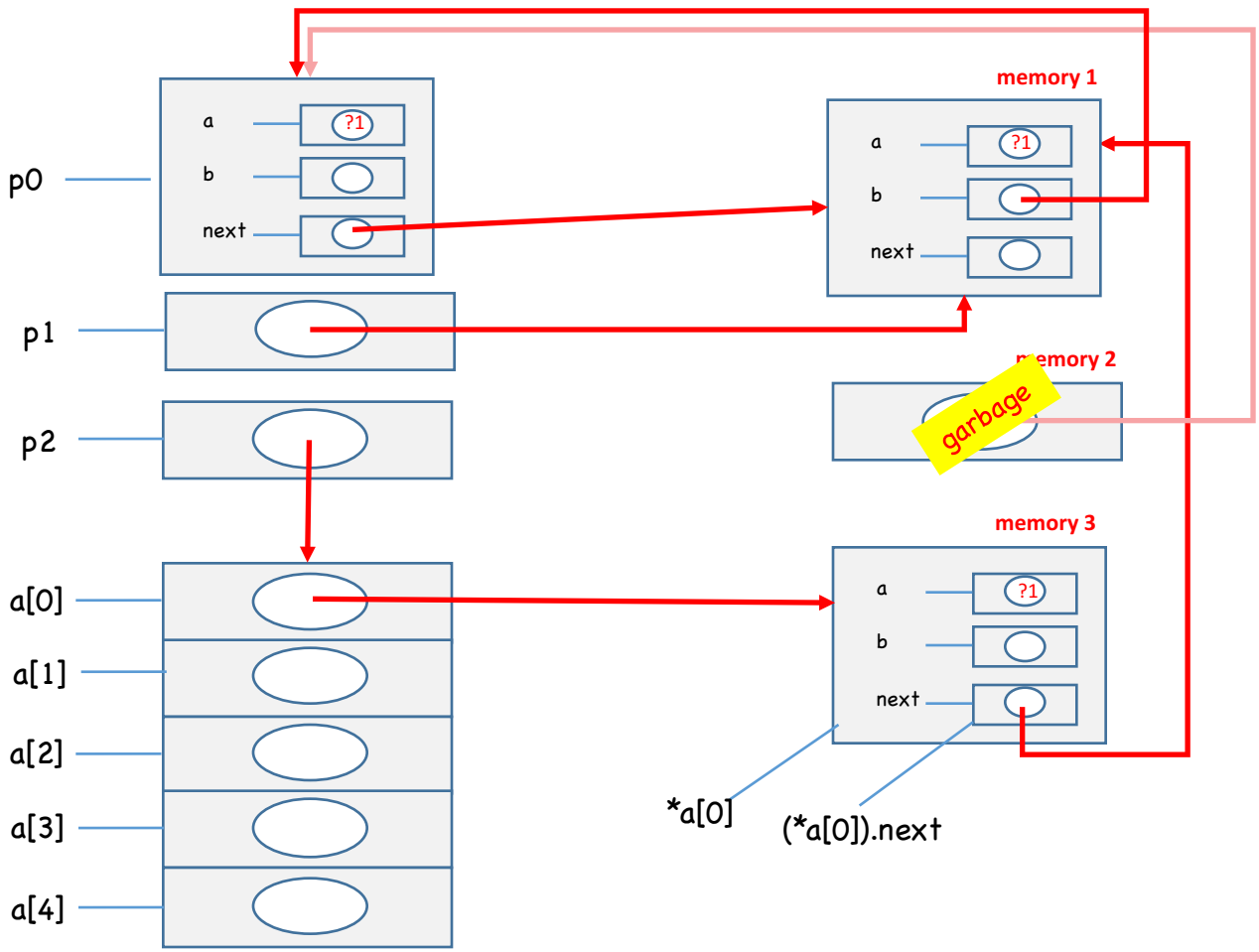
```
//-----------------------------------------------------------------
// Question1. Draw a box-circle diagram for p,q, p0, p1, p2 and the
//            memory allocated above at this point
//-----------------------------------------------------------------
```



Note: I use ?x to indicate unknown values. ?1 and ?2 can be different or the same, but all ?1 are the same.

Note: You do not need to show all variables in your solution. Only the memory allocated with malloc() and the specified variables

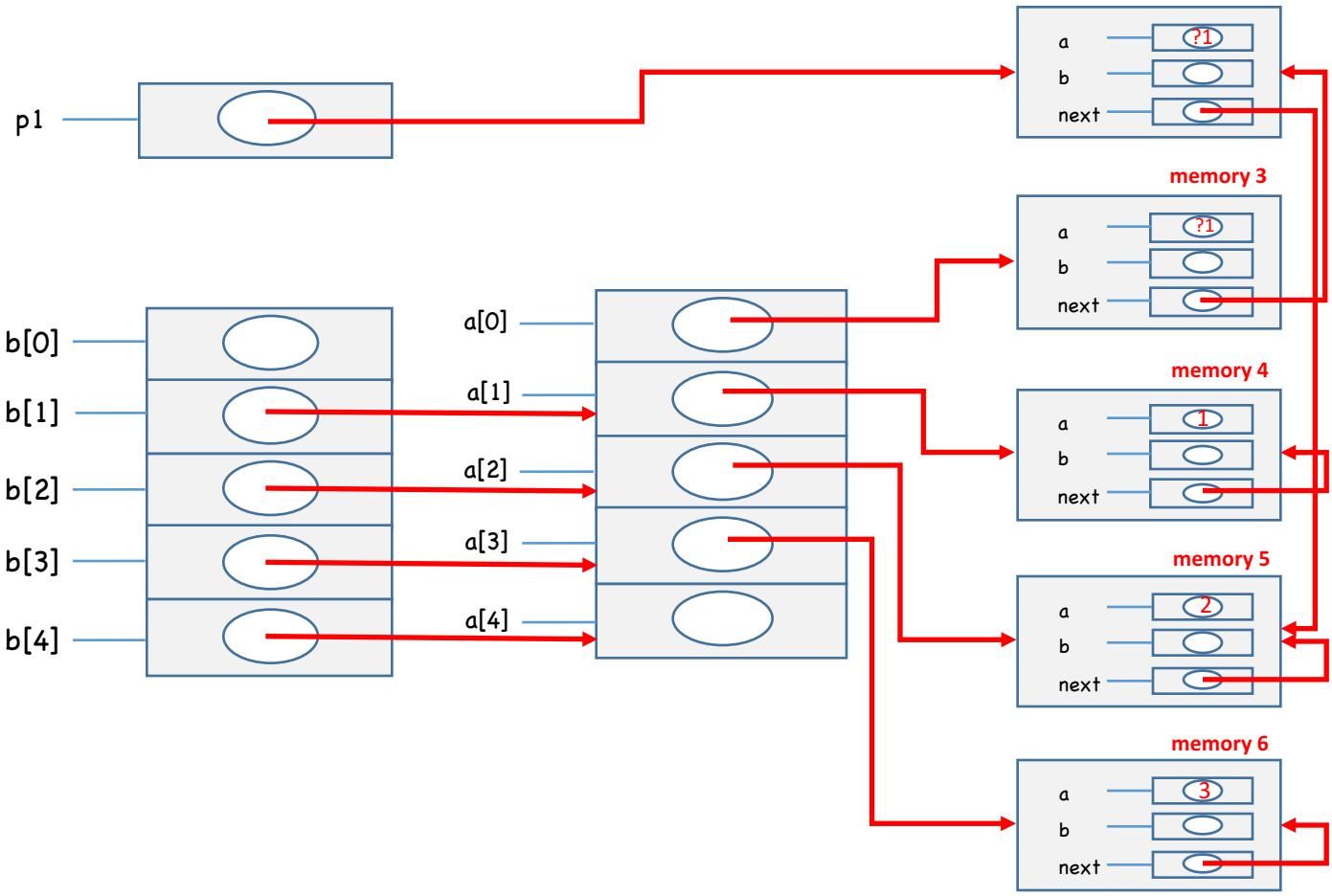
```
// scope 1
{   struct T *a[5];
    struct T **b[5];
    int i,j;

    a[0] = (struct T*) malloc(sizeof(struct T));
    p2 = &(a[0]);
    (*a[0]).next = p1;

    //-------------------------------------------------------------
    // Question 2. Draw a box-circle diagram for p1, p2, a[]
    //             and the memory allocated above at this point
    //-------------------------------------------------------------
```



Note a has no location associated with it. We cannot say a = ... but in C the expressions &a is the same as &a[0] and *a the same as a[0]

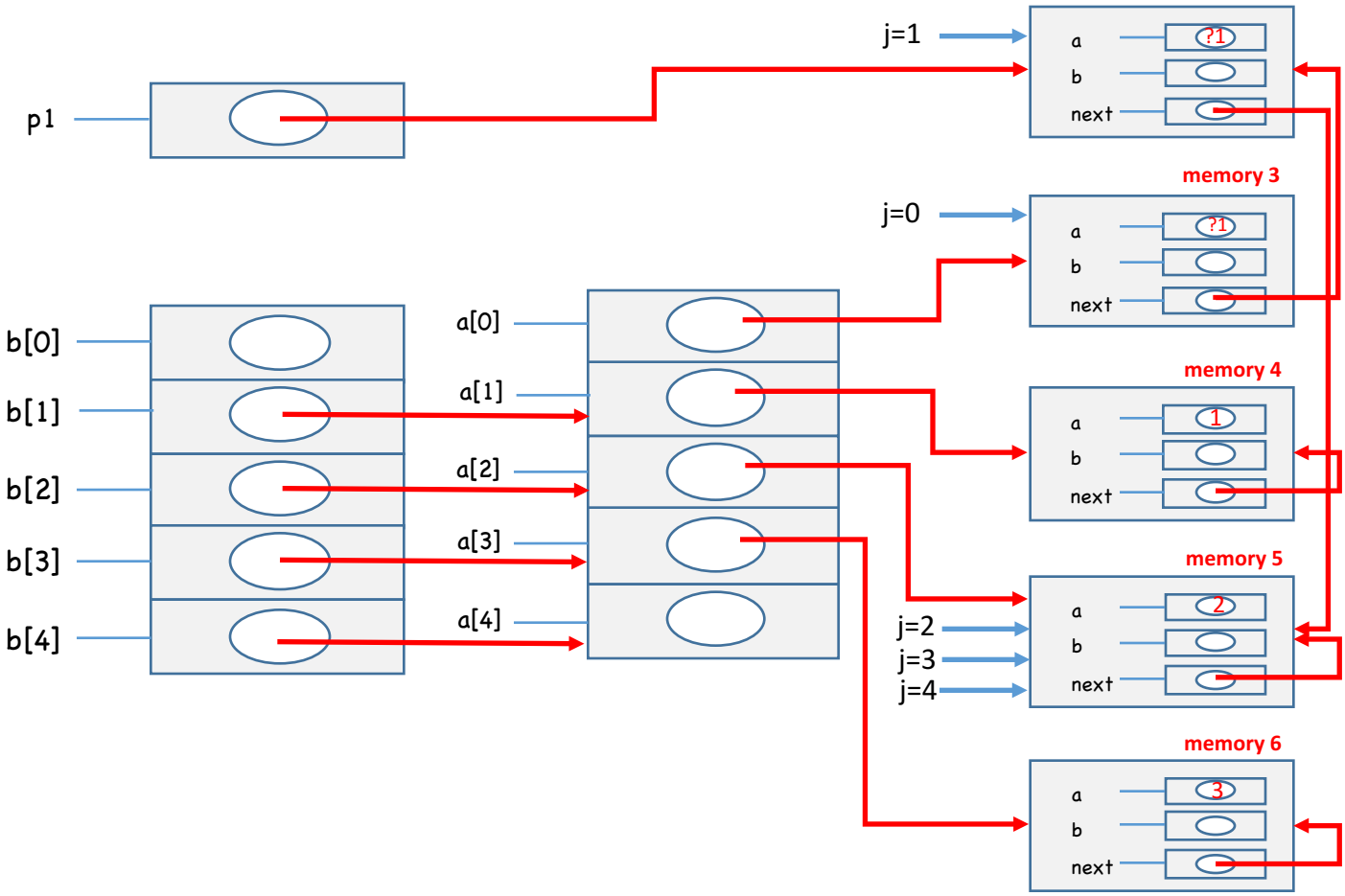
```
for (i = 1; i < 4; i++)
{
    a[i] = (struct T*) malloc(sizeof(struct T));
    (*a[i]).next = a[i-1];
    (*a[i]).a = i;
    b[i] = &a[i];
    (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```

After executing the code above, we get the following

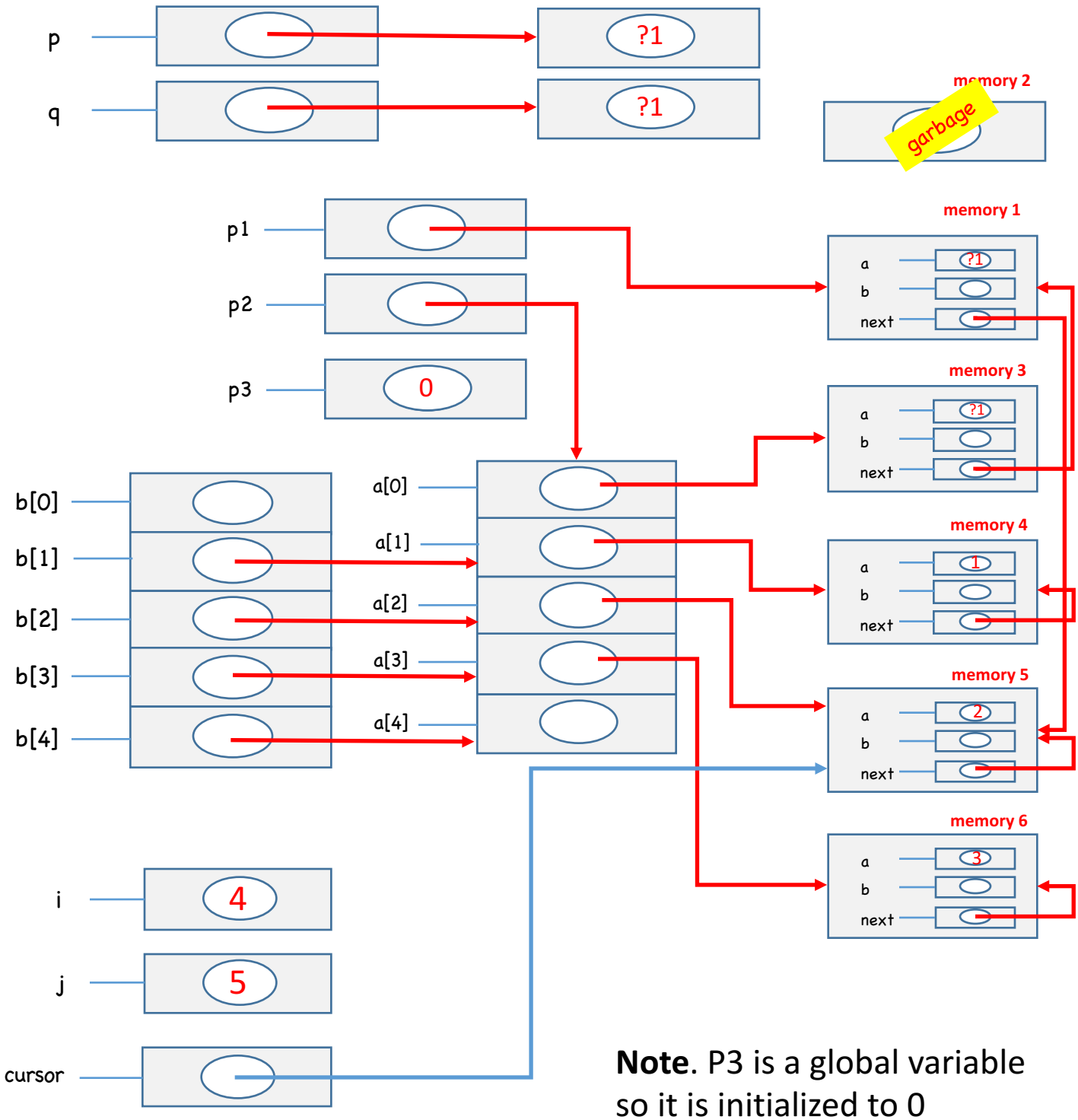


```
//-------------------------------------------------------------
// Question 3. Explain the output produced by the
//             following loop
//-------------------------------------------------------------
cursor = a[0];
for (j = 0; j < 5; j++)
{
    printf("%d  ",(*cursor).a);
    cursor = (*cursor).next;
}
printf("\n");
```

The following illustrate the value of cursor (blue arrow) for the successive values of j.
It also explains the values being printed which are ?1 ?1 2 2 2
In a particular execution ?1 might be 0, but in general it need not be 0



```
//-------------------------------------------------------------
// Question 4. Draw a box-circle diagram of ALL the variables
//             and the memory allocated up to this point?
//             what locations are garbage at this point?
//-------------------------------------------------------------
```

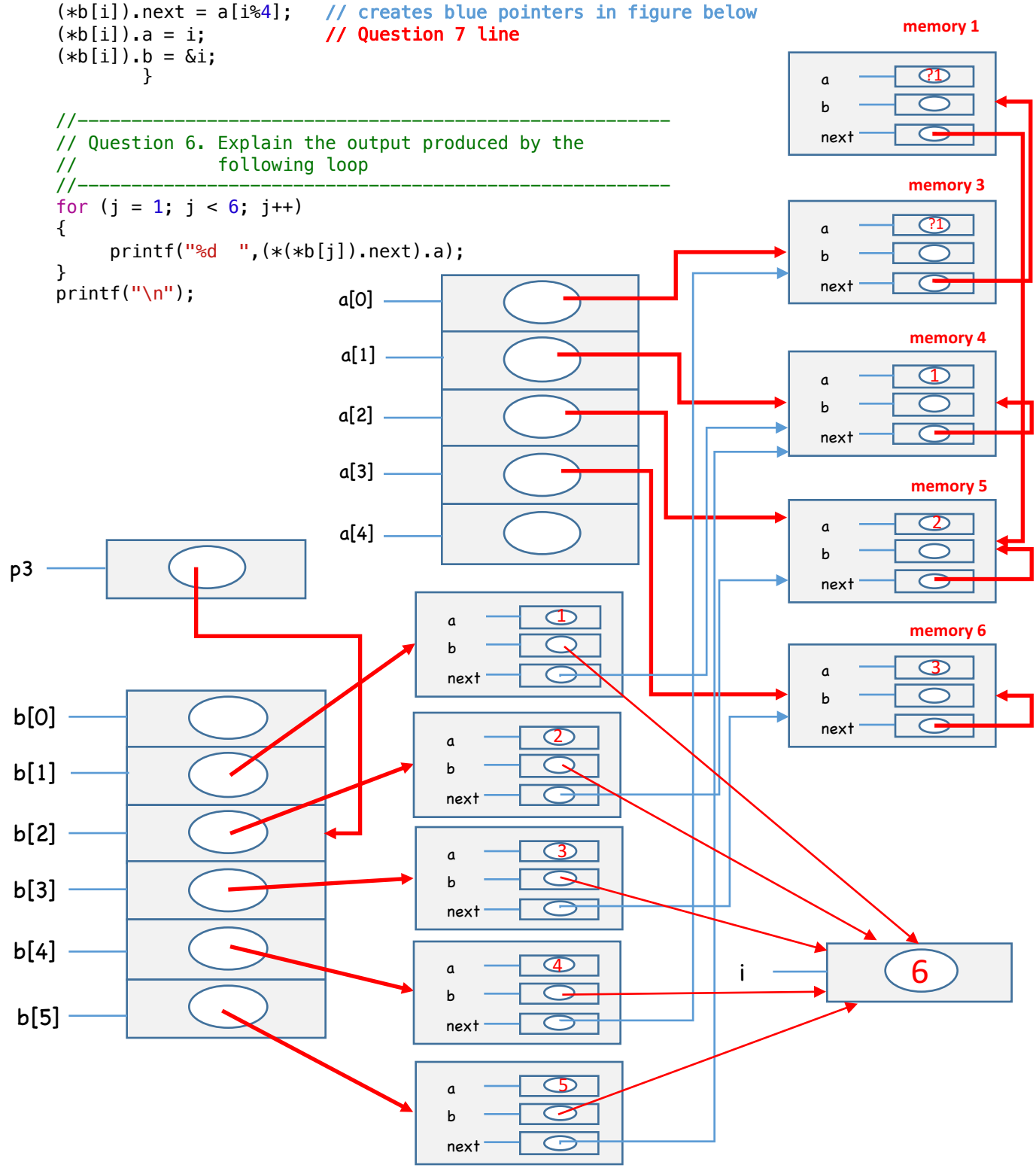


Note. P3 is a global variable so it is initialized to 0

```
//-------------------------------------------------------------
// Question 5. Give an alias for a[2] at this point.
//             This alias should not include the names "a" !!
//             This might be tricky and you might want to skip
//             it and do other parts first.
//-------------------------------------------------------------
```

Answer: *(p2+2) or p2[2] or *(b[2])

```
// scope 2
{   struct T *b[6];
    int j;

    p3 = &b[2];
    for (i = 1; i < 6; i++)
    {
        b[i] = (struct T*) malloc(sizeof(struct T));
        (*b[i]).next = a[i%4];    // creates blue pointers in figure below
        (*b[i]).a = i;            // Question 7 line
        (*b[i]).b = &i;
                }

    //-------------------------------------------------------
    // Question 6. Explain the output produced by the
    //             following loop
    //-------------------------------------------------------
    for (j = 1; j < 6; j++)
    {
         printf("%d  ",(*(*b[j]).next).a);
    }
    printf("\n");
```



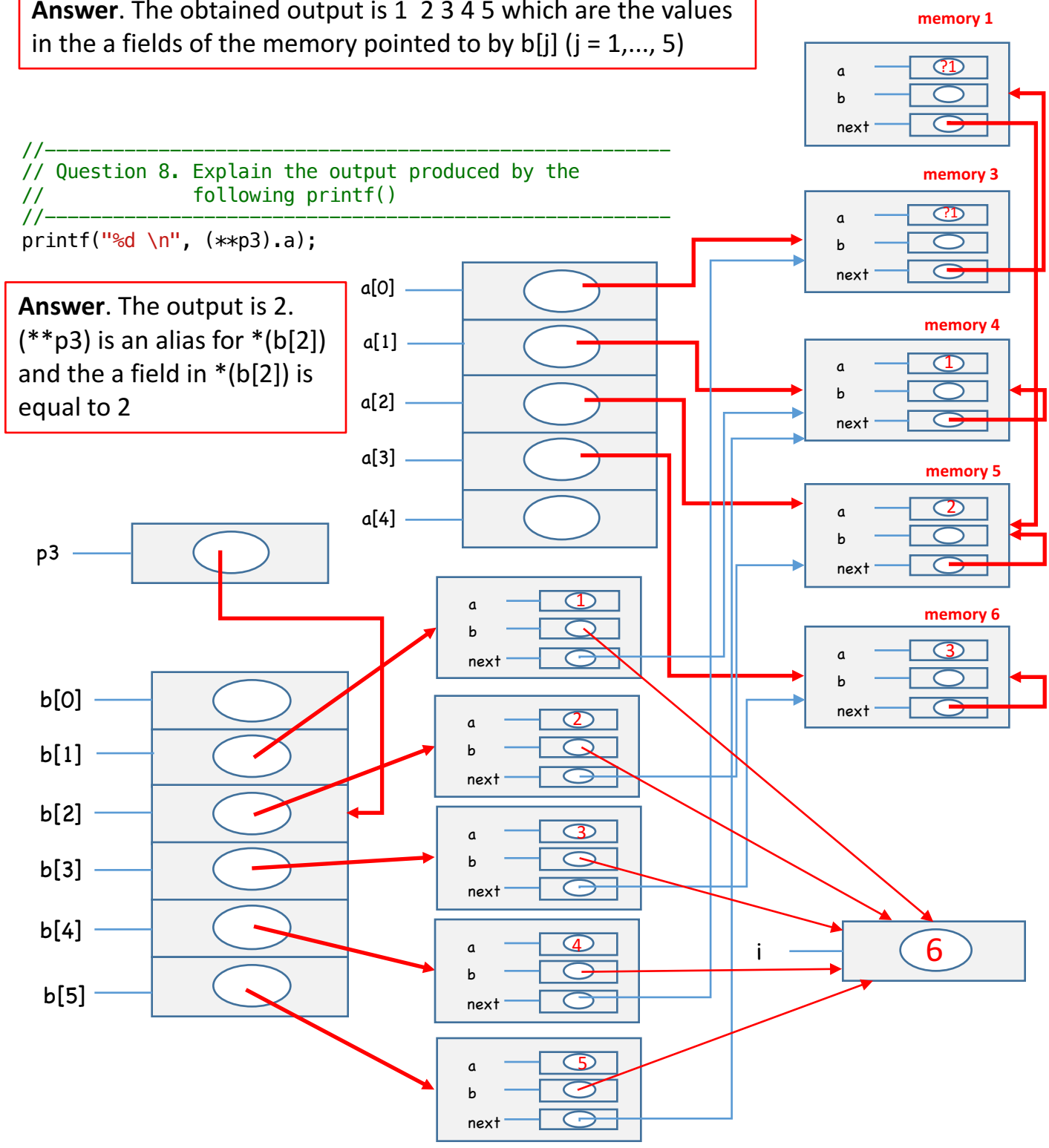
The output is 1 2 3 0 1 which is *a[1].a *a[2].a *a[3].a *a[0].a , and *a[1].a
The value 0 just happens to be in the field *a[0].a. The other values have been set by the program

```
//-------------------------------------------------------
// Question 7. Explain the output produced by the
//             following loop
//-------------------------------------------------------
for (j = 1; j < 6; j++)
{
    printf("%d  ",(*b[j]).a);
}
```

Answer. The obtained output is 1 2 3 4 5 which are the values in the a fields of the memory pointed to by b[j] (j = 1,..., 5)

```
//-------------------------------------------------------
// Question 8. Explain the output produced by the
//             following printf()
//-------------------------------------------------------
printf("%d \n", (**p3).a);
```

Answer. The output is 2. (**p3) is an alias for *(b[2]) and the a field in *(b[2]) is equal to 2

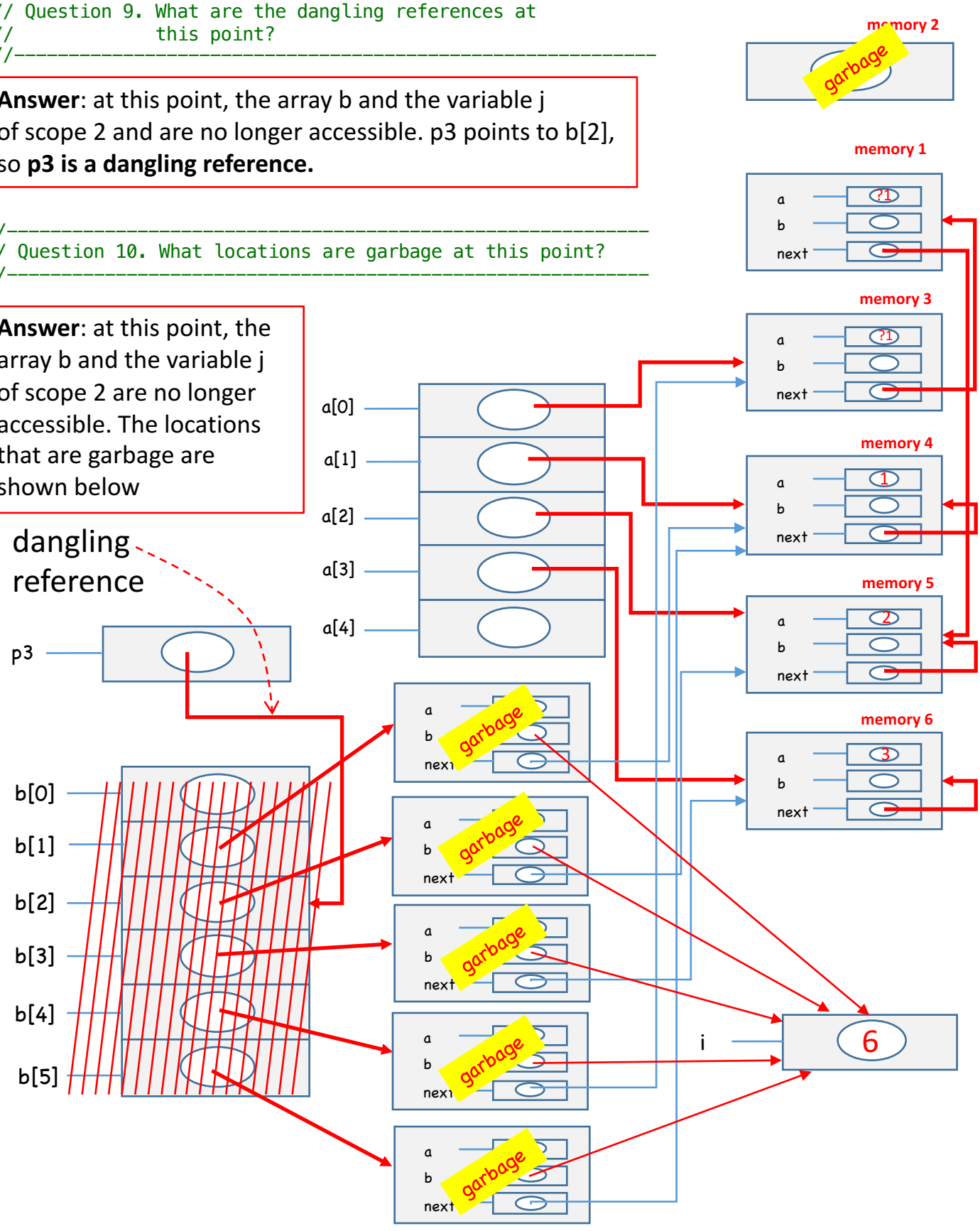


```
//-------------------------------------------------------------
// Question 9. What are the dangling references at
//             this point?
//-------------------------------------------------------------
```

Answer: at this point, the array b and the variable j of scope 2 and are no longer accessible. p3 points to b[2], so **p3 is a dangling reference.**

```
//-------------------------------------------------------------
// Question 10. What locations are garbage at this point?
//-------------------------------------------------------------
```

Answer: at this point, the array b and the variable j of scope 2 are no longer accessible. The locations that are garbage are shown below



```
//-------------------------------------------------------------------
// Question 11. What are the dangling references at this point.
//-------------------------------------------------------------------
```

Answer: at this point, the locations for the array a are deallocated, but the locations are still accessible. p2 is a dangling reference. Also, p3 is still a dangling reference at this point