CSE 340 FALL 2021

Pointer Semantics
Step by Step Example

Prepared by Rida Bazzi

Pointer semantics.

We will examine a complicated C program and analyze the pointer operations

Pointer semantics.

We will examine a complicated C program and analyze the pointer operations

The program is an actual C program that you can compile and run

Pointer semantics.

We will examine a complicated C program and analyze the pointer operations

The program is an actual C program that you can compile and run

We will use the C99 semantics in analyzing the program

Pointer semantics.

We will examine a complicated C program and analyze the pointer operations

The program is an actual C program that you can compile and run

We will use the C99 semantics in analyzing the program

At each step, we will draw the box-circle diagrams to illustrate the effects of the operations

Pointer semantics.

We will examine a complicated C program and analyze the pointer operations

The program is an actual C program that you can compile and run

We will use the C99 semantics in analyzing the program

At each step, we will draw the box-circle diagrams to illustrate the effects of the operations

We start with the declarations section of the program

Declarations

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

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

struct T p0;

struct T *p1;
struct T **p2;

int *p;
int *q;
```

Declarations

```
#include <stdio.h>     // standard input/output
#include <stdlib.h>    // standard library

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

struct T p0;

struct T *p1;
struct T **p2;

int *p;
int *q;
```

Declarations

```
#include <stdio.h>      // standard input/output
#include <stdlib.h>     // standard library

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

struct T p0;

struct T *p1;
struct T **p2;

int *p;
int *q;
```

Declarations

```
#include <stdio.h>      // standard input/output
#include <stdlib.h>     // standard library

struct T {              // structure T
       int a;           // integer field
       int *b;
       struct T *next;
} ;

struct T p0;

struct T *p1;
struct T **p2;

int *p;
int *q;
```

Declarations

```
#include <stdio.h>     // standard input/output
#include <stdlib.h>    // standard library

struct T {             // structure T
       int a;          // integer field
       int *b;         // pointer to int
       struct T *next;
} ;

struct T p0;

struct T *p1;
struct T **p2;

int *p;
int *q;
```

Declarations

```
#include <stdio.h>     // standard input/output
#include <stdlib.h>    // standard library

struct T {             // structure T
       int a;          // integer field
       int *b;         // pointer to int
       struct T *next; // pointer to struct T
} ;

struct T p0;

struct T *p1;
struct T **p2;

int *p;
int *q;
```

Declarations

```
#include <stdio.h>     // standard input/output
#include <stdlib.h>    // standard library

struct T {             // structure T
       int a;          // integer field
       int *b;         // pointer to int
       struct T *next; // pointer to struct T
} ;

struct T p0;           // p0’s type is struct T
                       // (not pointer)
struct T *p1;
struct T **p2;

int *p;
int *q;
```

Declarations

```
#include <stdio.h>       // standard input/output
#include <stdlib.h>      // standard library

struct T {               // structure T
       int a;            // integer field
       int *b;           // pointer to int
       struct T *next;   // pointer to struct T
} ;

struct T p0;             // p0's type is struct T
                         // (not pointer)
struct T *p1;            // pointer to struct T
struct T **p2;

int *p;
int *q;
```

Declarations

```
#include <stdio.h>     // standard input/output
#include <stdlib.h>    // standard library

struct T {             // structure T
       int a;          // integer field
       int *b;         // pointer to int
       struct T *next; // pointer to struct T
} ;

struct T p0;           // p0’s type is struct T
                       // (not pointer)
struct T *p1;          // pointer to struct T
struct T **p2;         // pointer to struct T *

int *p;
int *q;
```

Declarations

```
#include <stdio.h>     // standard input/output
#include <stdlib.h>    // standard library

struct T {             // structure T
       int a;          // integer field
       int *b;         // pointer to int
       struct T *next; // pointer to struct T
} ;

struct T p0;           // p0's type is struct T
                       // (not pointer)
struct T *p1;          // pointer to struct T
struct T **p2;         // pointer to struct T *

int *p;                // pointer to int
int *q;                // pointer to int
```

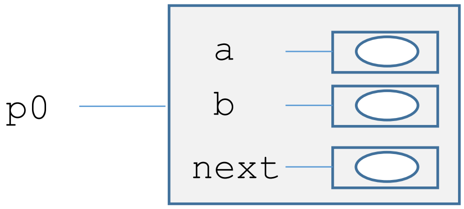
Declarations

```
#include <stdio.h>       // standard input/output
#include <stdlib.h>      // standard library

struct T {               // structure T
       int a;            // integer field
       int *b;           // pointer to int
       struct T *next;   // pointer to struct T
} ;

struct T p0;             // p0’s type is struct T
                         // (not pointer)
struct T *p1;            // pointer to struct T
struct T **p2;           // pointer to struct T *

int *p;                  // pointer to int
int *q;                  // pointer to int
```

Let us draw the box-circle diagram at this point

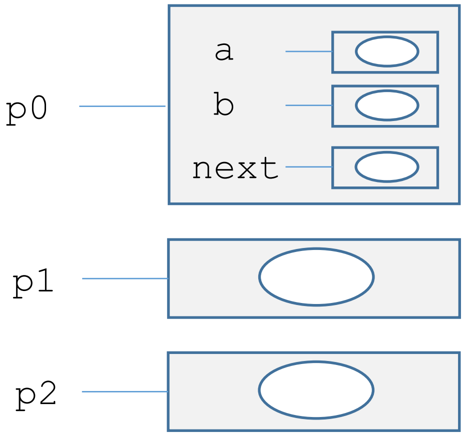
Declarations

```
#include <stdio.h>       // standard input/output
#include <stdlib.h>      // standard library

struct T {               // structure T
       int a;            // integer field
       int *b;           // pointer to int
       struct T *next;   // pointer to struct T
} ;

struct T p0;             // p0's type is struct T
                         // (not pointer)
struct T *p1;            // pointer to struct T
struct T **p2;           // pointer to struct T *

int *p;                  // pointer to int
int *q;                  // pointer to int
```



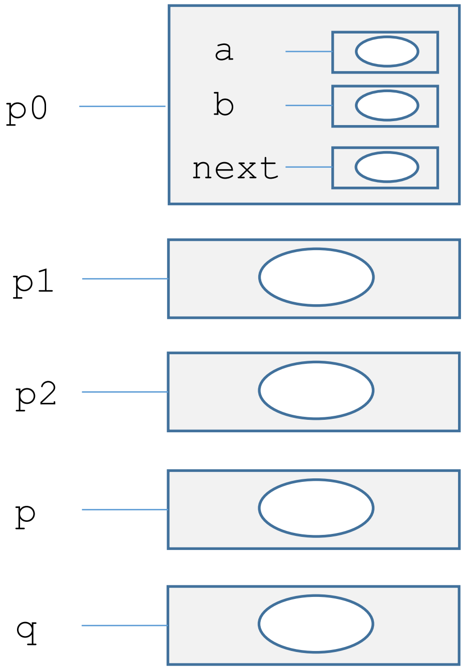
Declarations

```
#include <stdio.h>       // standard input/output
#include <stdlib.h>      // standard library

struct T {               // structure T
       int a;            // integer field
       int *b;           // pointer to int
       struct T *next;   // pointer to struct T
} ;

struct T p0;             // p0's type is struct T
                         // (not pointer)
struct T *p1;            // pointer to struct T
struct T **p2;          // pointer to struct T *

int *p;                  // pointer to int
int *q;                  // pointer to int
```



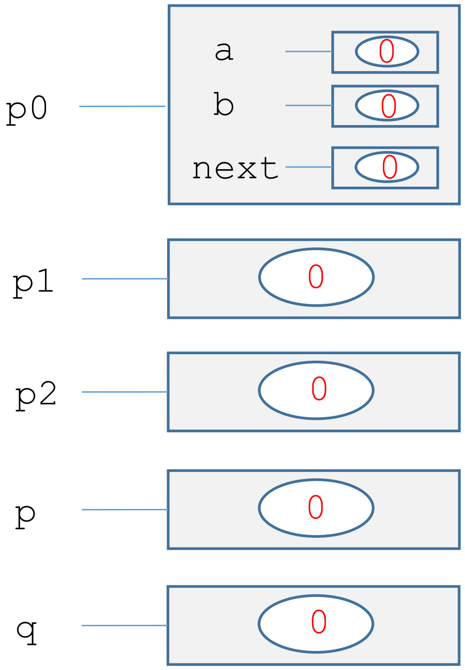
Declarations

```
#include <stdio.h>       // standard input/output
#include <stdlib.h>      // standard library

struct T {               // structure T
       int a;            // integer field
       int *b;           // pointer to int
       struct T *next;   // pointer to struct T
} ;

struct T p0;             // p0's type is struct T
                         // (not pointer)
struct T *p1;            // pointer to struct T
struct T **p2;           // pointer to struct T *

int *p;                  // pointer to int
int *q;                  // pointer to int
```



Declarations

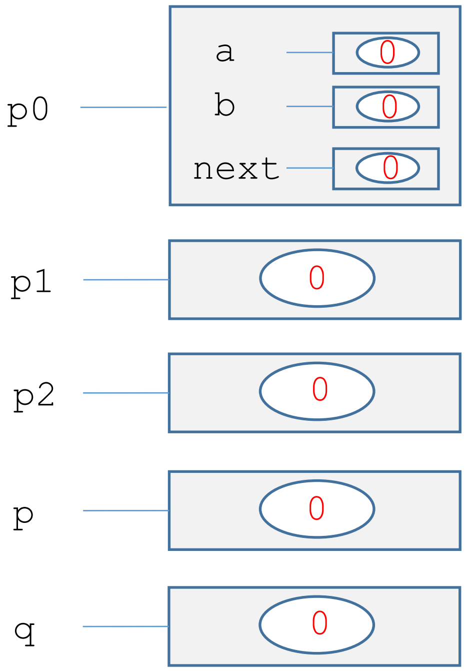
```
#include <stdio.h>      // standard input/output
#include <stdlib.h>     // standard library

struct T {              // structure T
       int a;           // integer field
       int *b;          // pointer to int
       struct T *next;  // pointer to struct T
} ;

struct T p0;            // p0's type is struct T
                        // (not pointer)
struct T *p1;           // pointer to struct T
struct T **p2;          // pointer to struct T *

int *p;                 // pointer to int
int *q;                 // pointer to int
```



All values of global variables are initially 0 according to the C standard

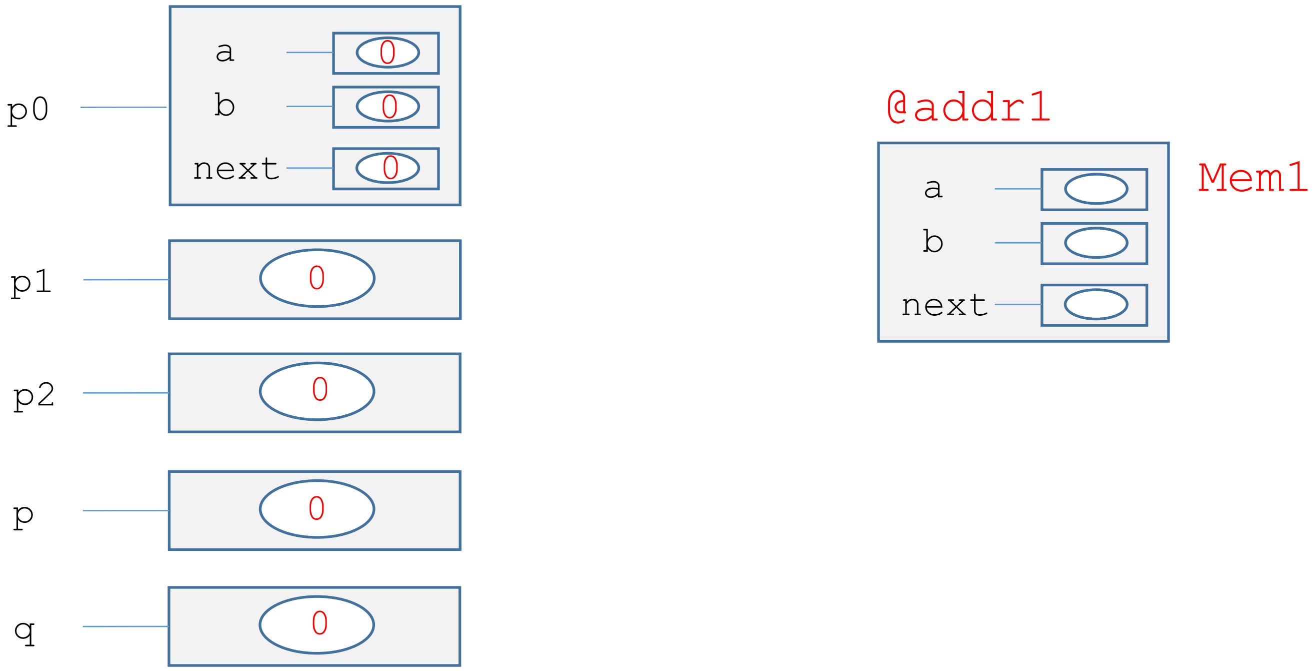
main()

```
int main()
{
 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;
```



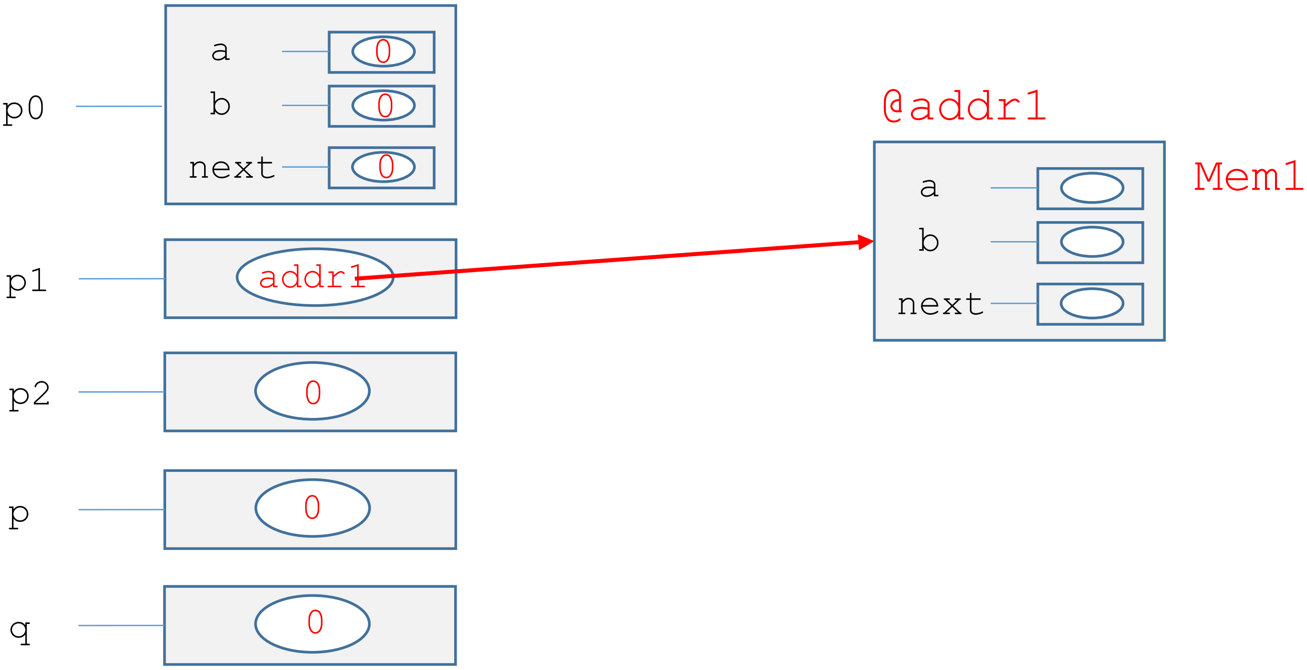
main()

```
int main()
{
 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;
```



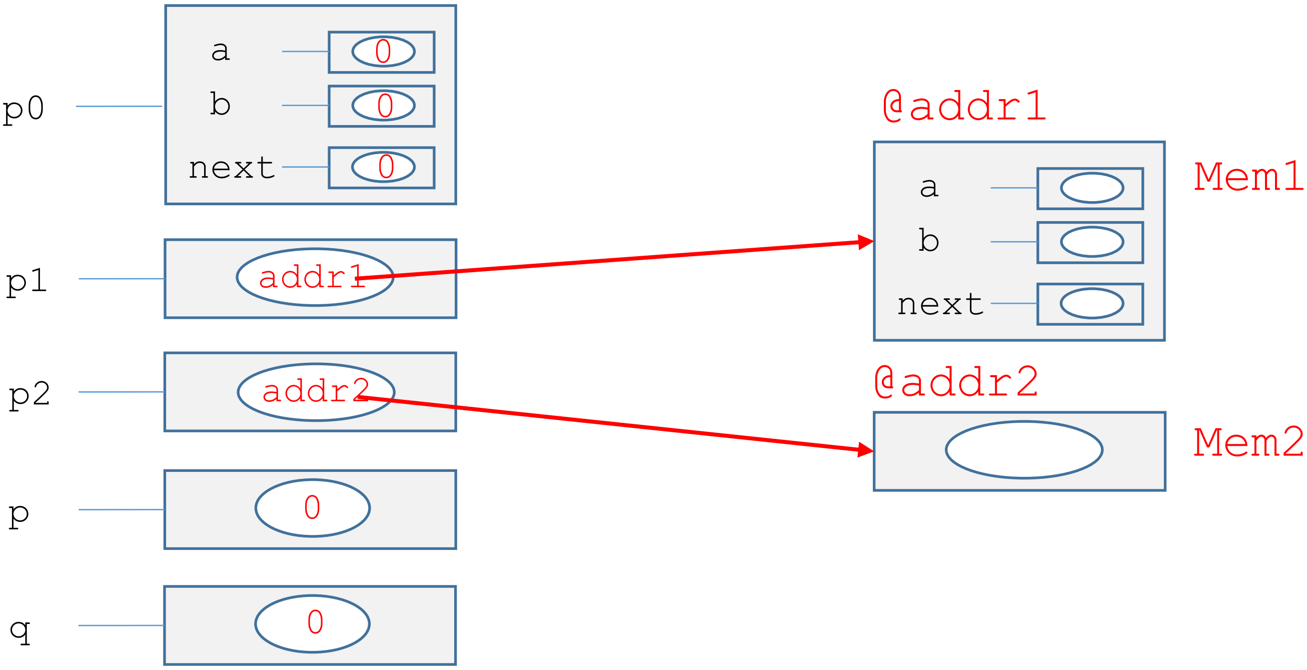
```
int main()
{
 p1 = (struct T *) malloc(sizeof(struct T));
 p2 = (struct T **) malloc(sizeof(struct T *));

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

 *p = *q;
```

main()



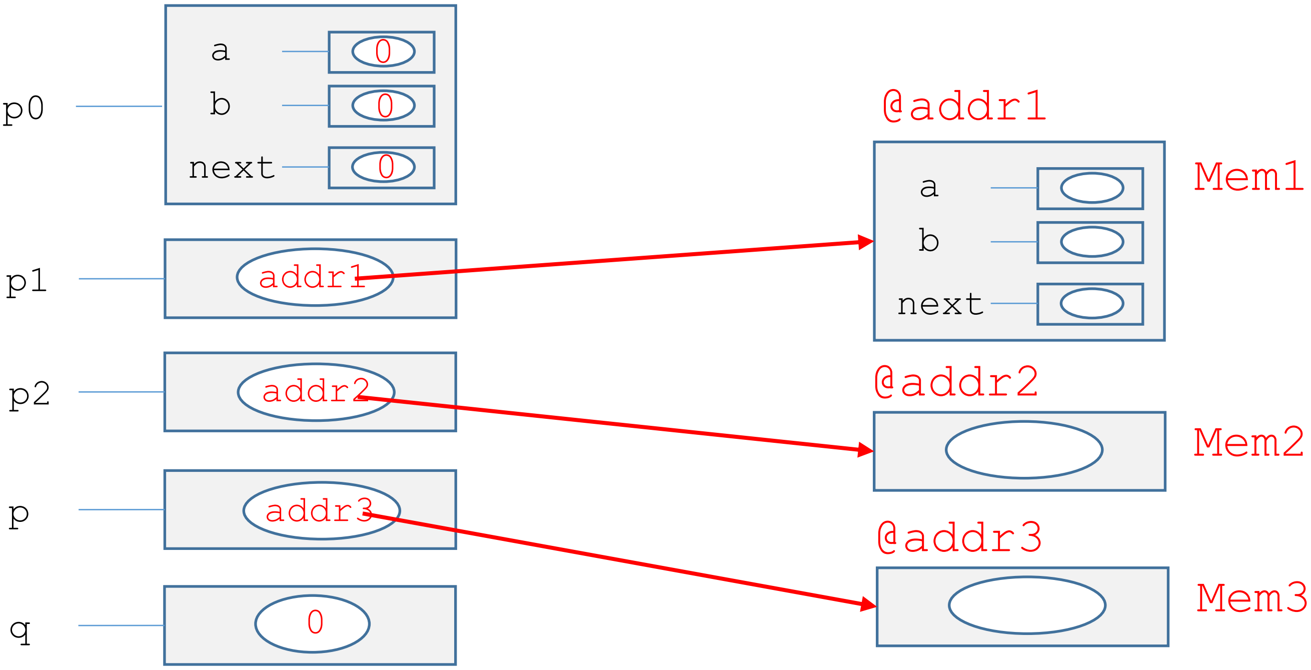
```
int main()
{
 p1 = (struct T *) malloc(sizeof(struct T));
 p2 = (struct T **) malloc(sizeof(struct T *));

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

 *p = *q;
```

main()



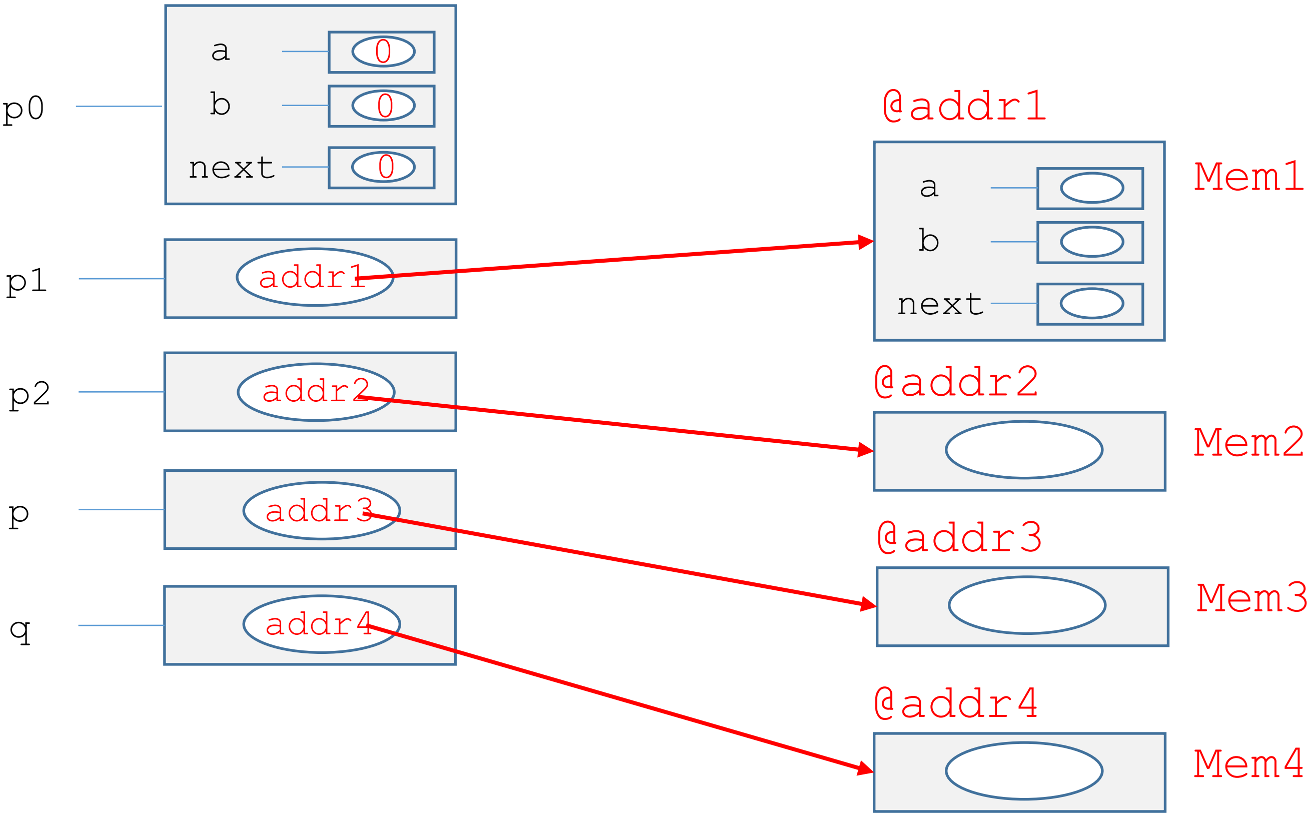
main()

```
int main()
{
 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;
```



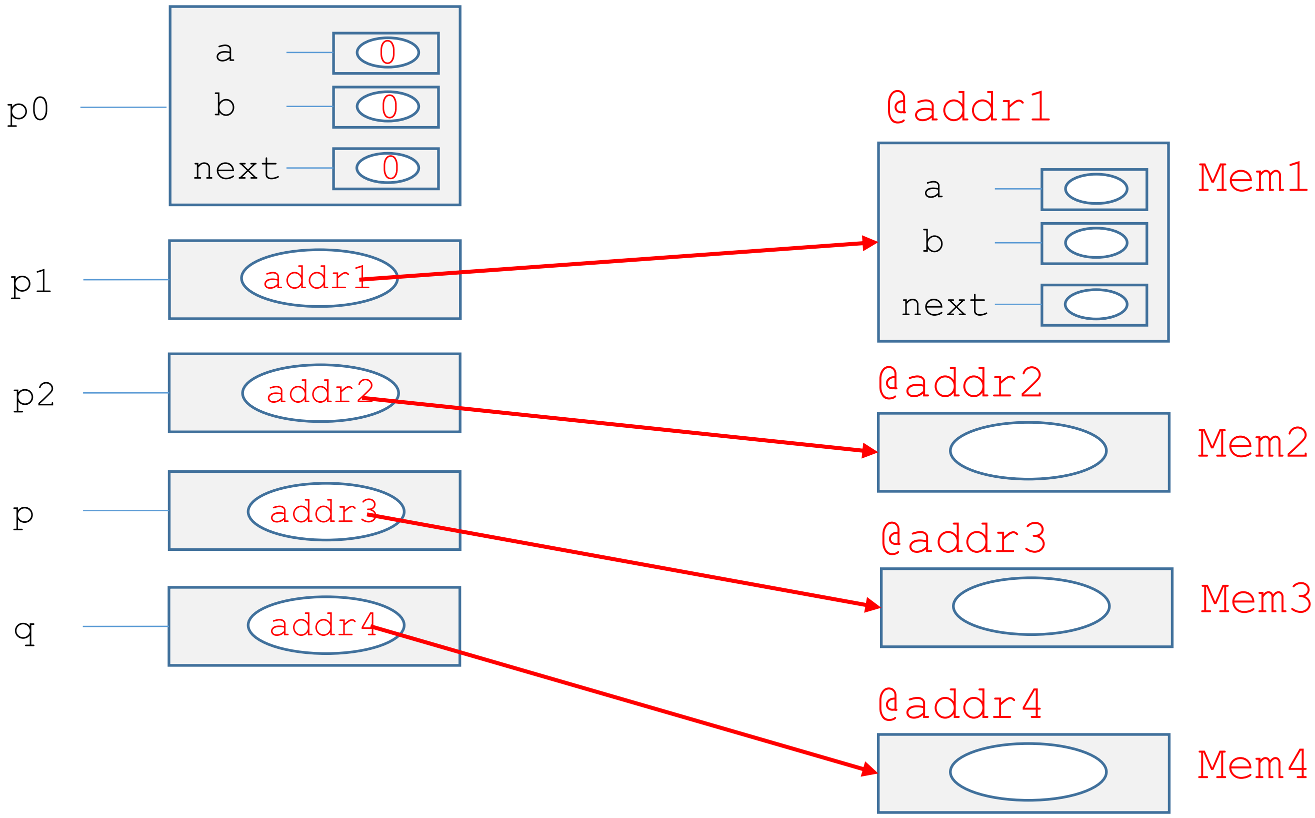
main()

```
int main()
{
 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;
```



```
int main()
{
 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;
```

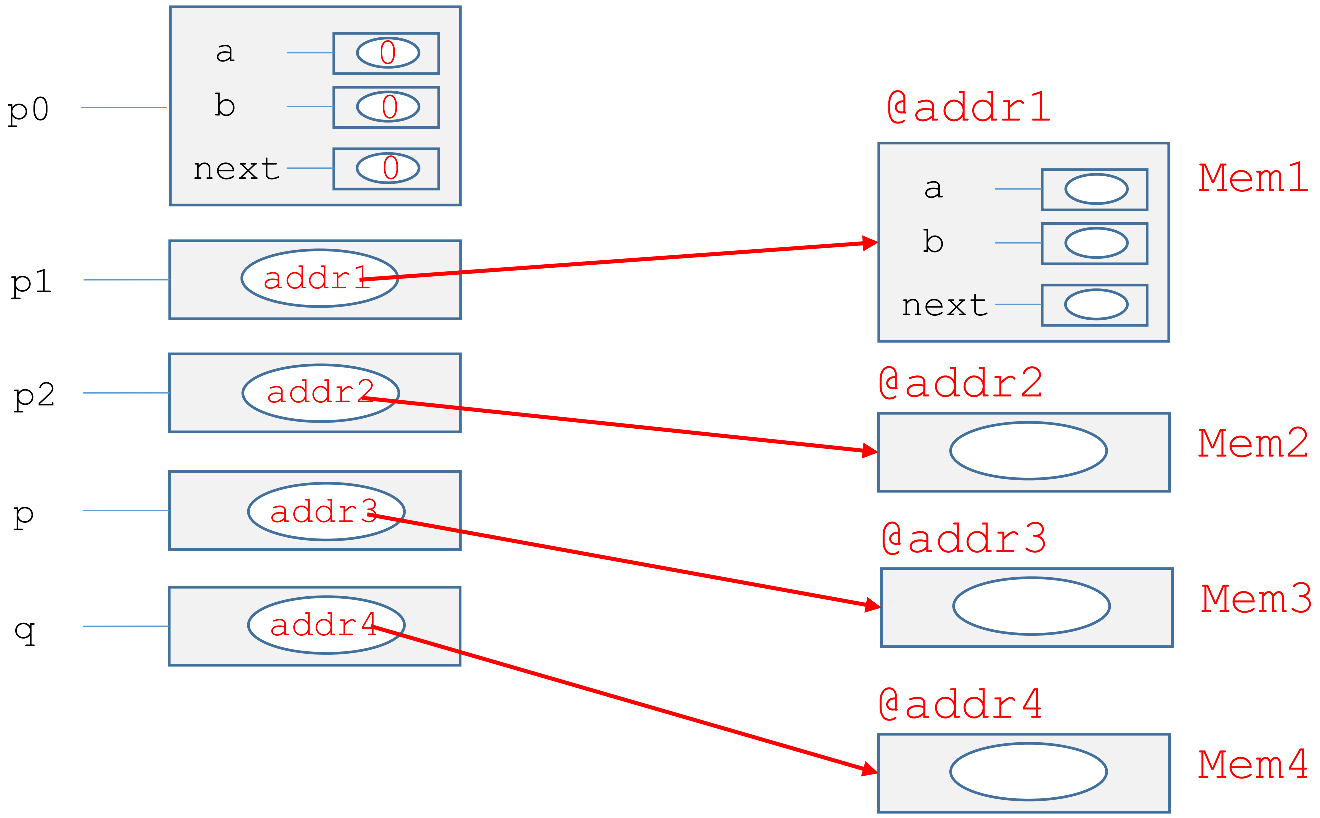
main()



The values in the allocated locations are not initialized. So, I will denote them with ?#

```
int main()
{
 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;
```

main()

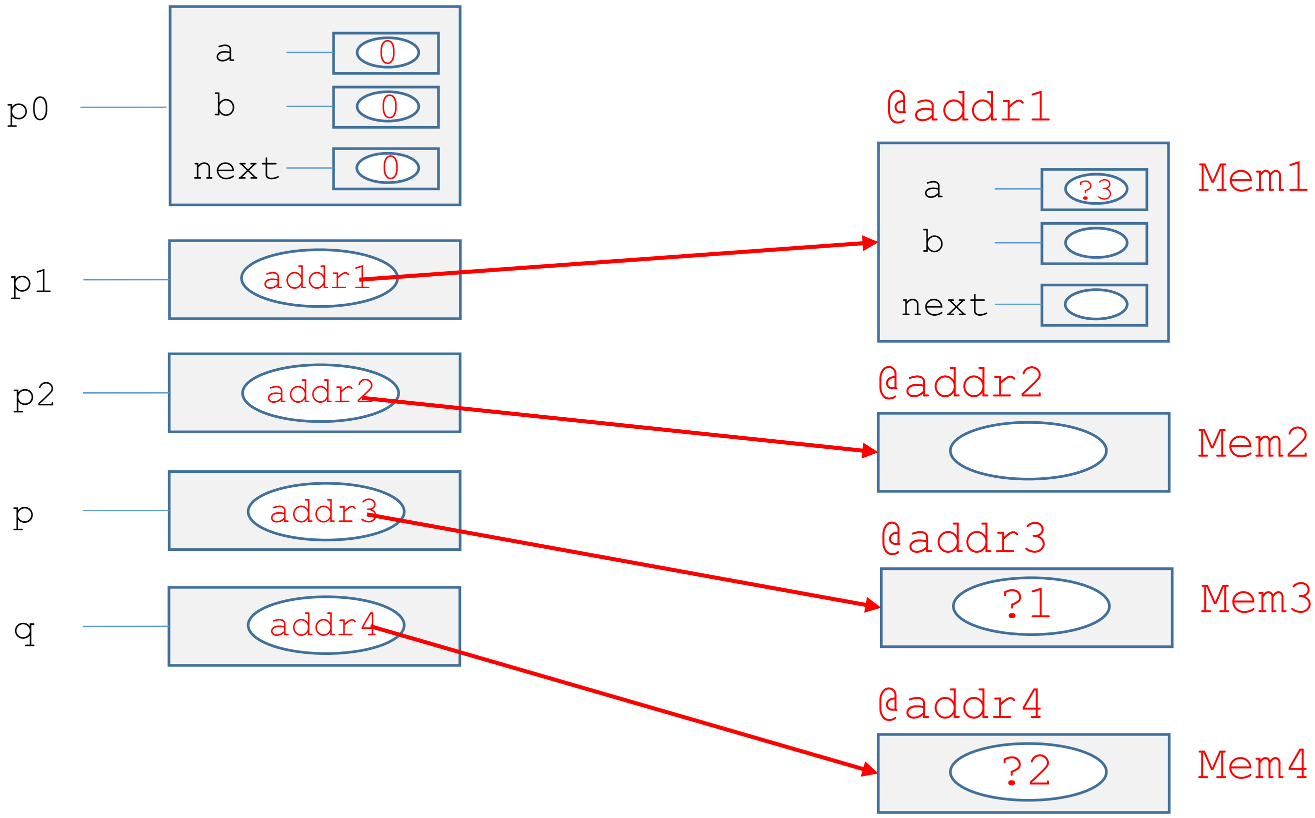


The values in the allocated locations are not initialized. So, I will denote them with ?#

The ? to indicate the value is unknown

main()

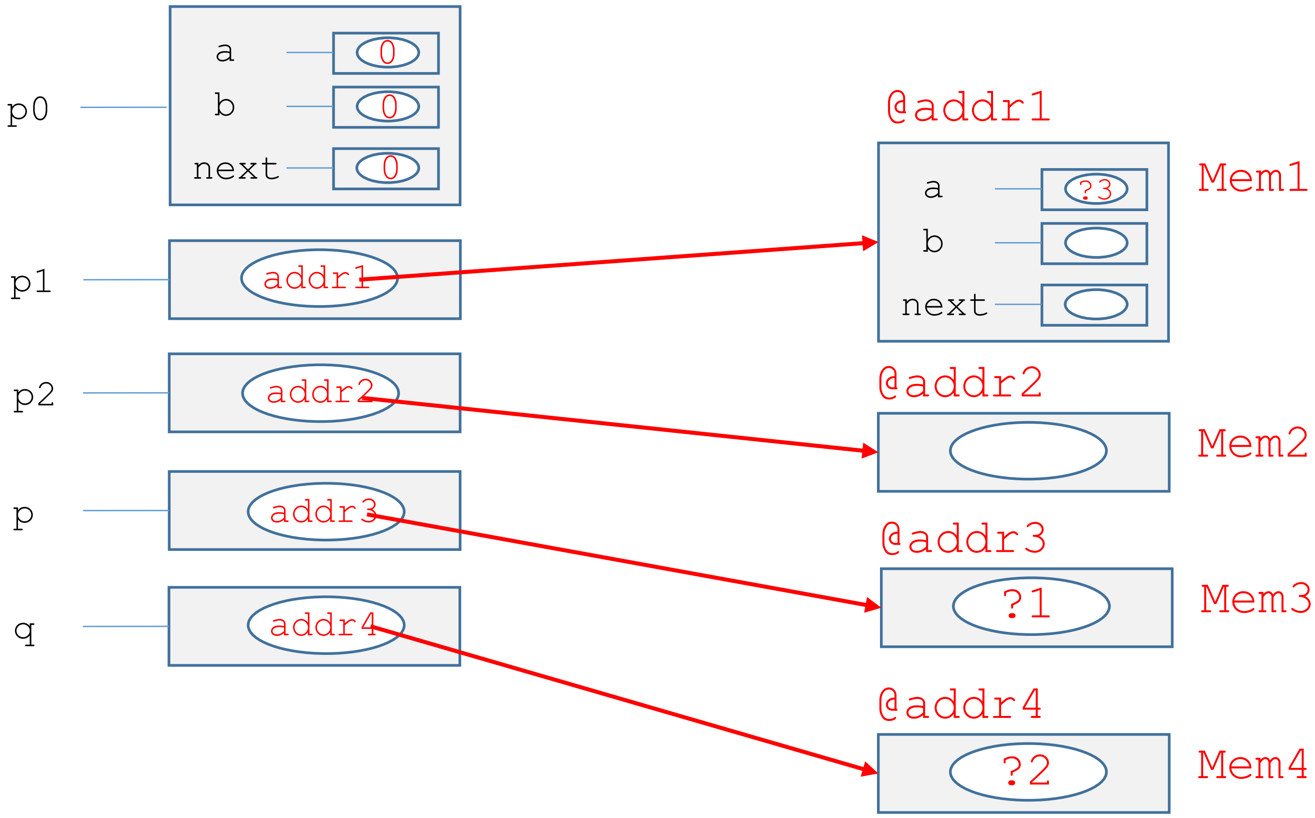
```
int main()
{
 p1 = (struct T *) malloc(sizeof(struct T));
 p2 = (struct T **) malloc(sizeof(struct T *));

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

 *p = *q;
```



Above, 3 of the unknown value are highlighted

```
int main()                                        
{
 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;
```

main()

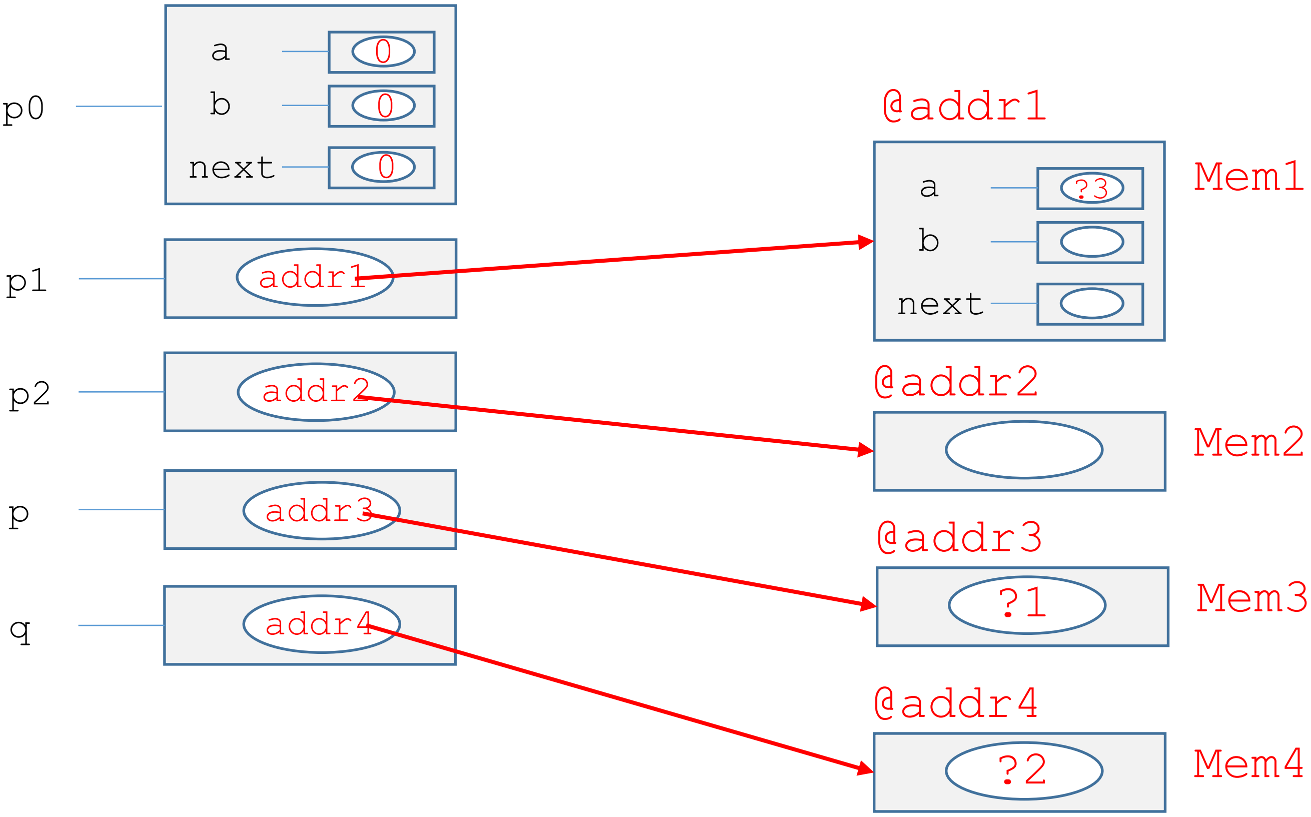


Above, 3 of the unknown value are highlighted

They are given different numbers because these values need not be the same

main()

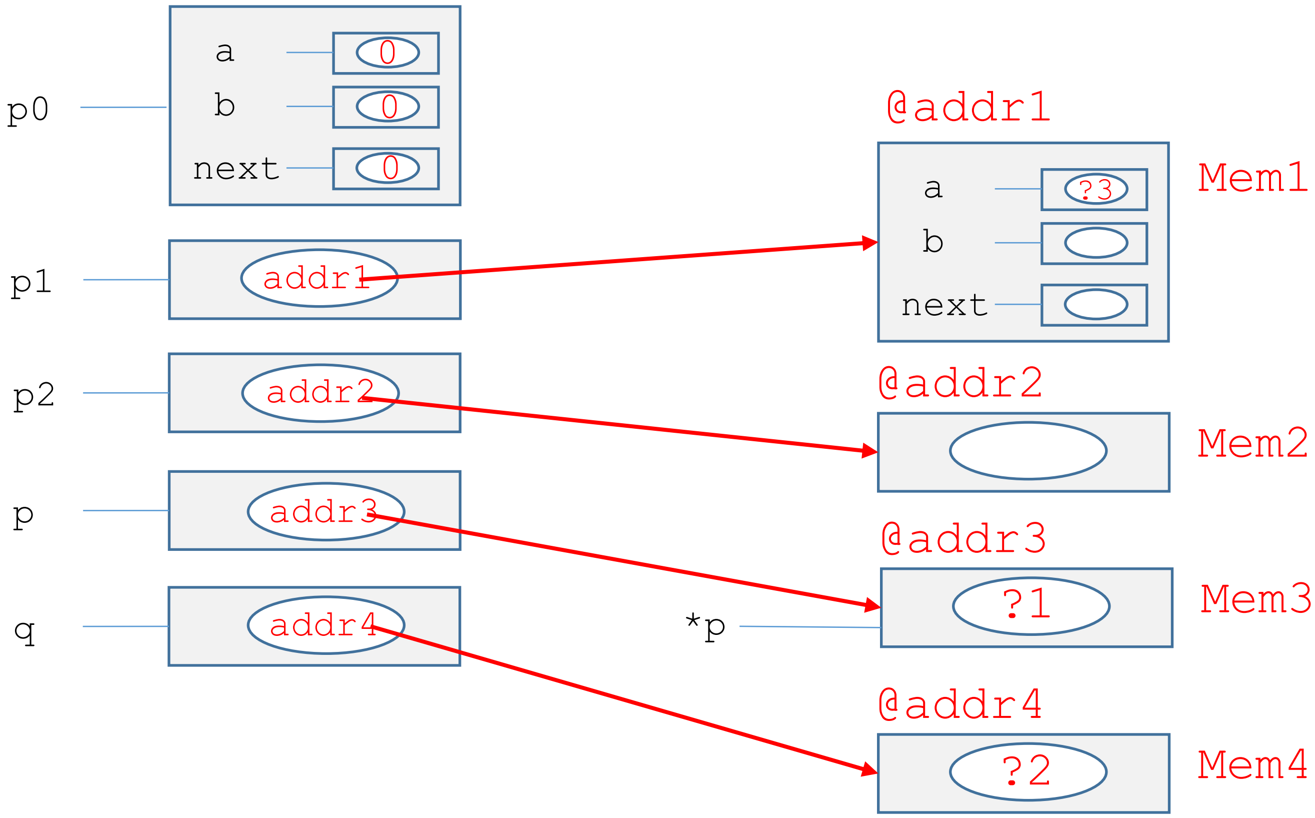
```
int main()
{
 p1 = (struct T *) malloc(sizeof(struct T));
 p2 = (struct T **) malloc(sizeof(struct T *));

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

 *p = *q;
```



Now, we are ready to go back to our execution

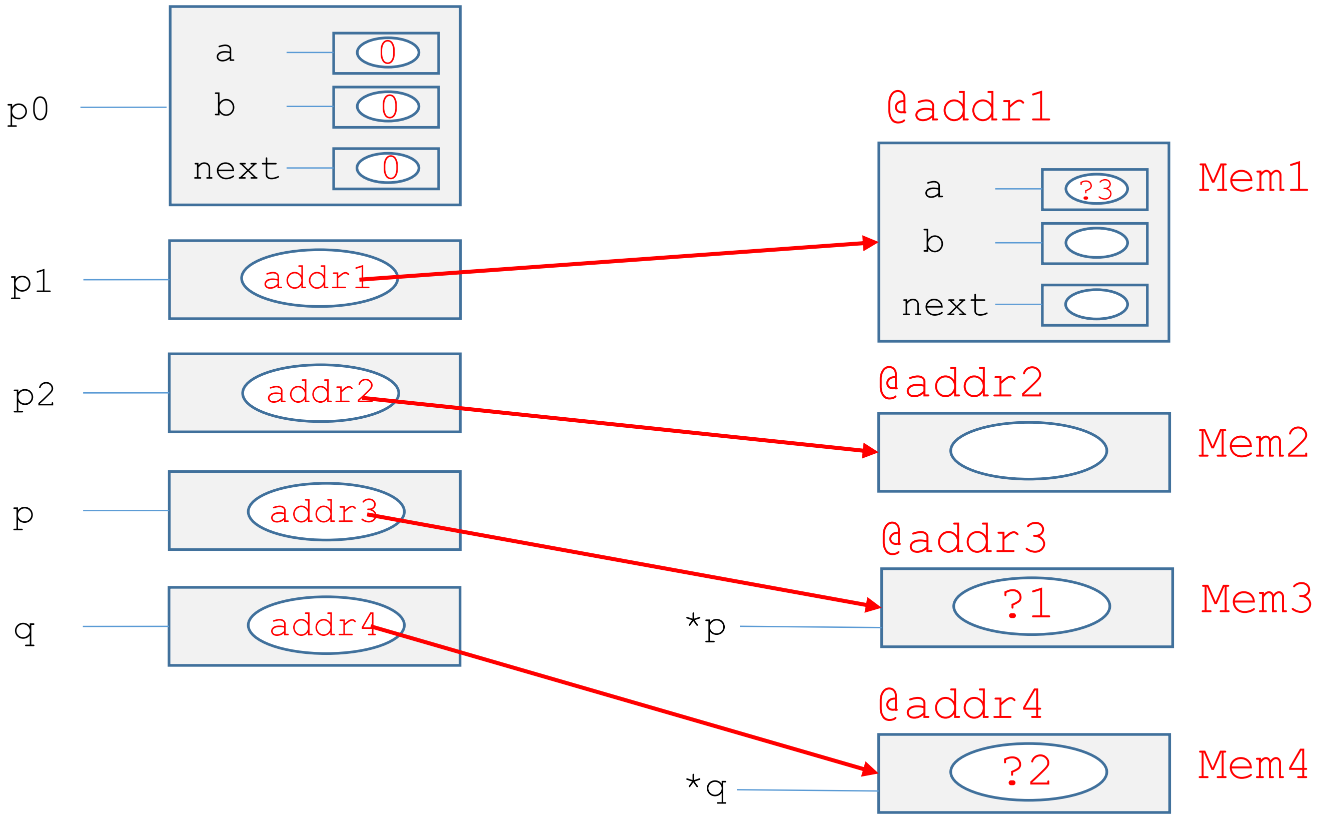
main()

```
int main()
{
 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;
```



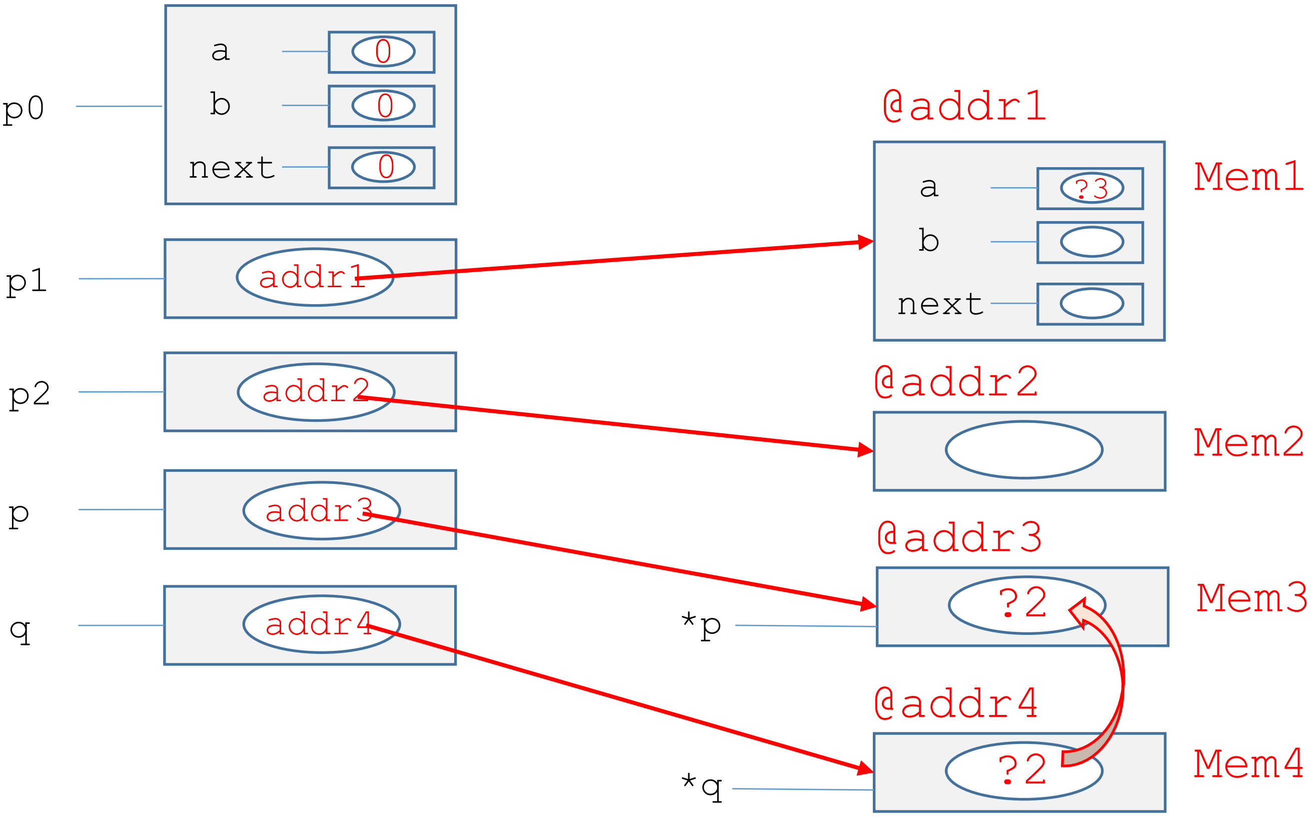
main()

```
int main()
{
 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;
```



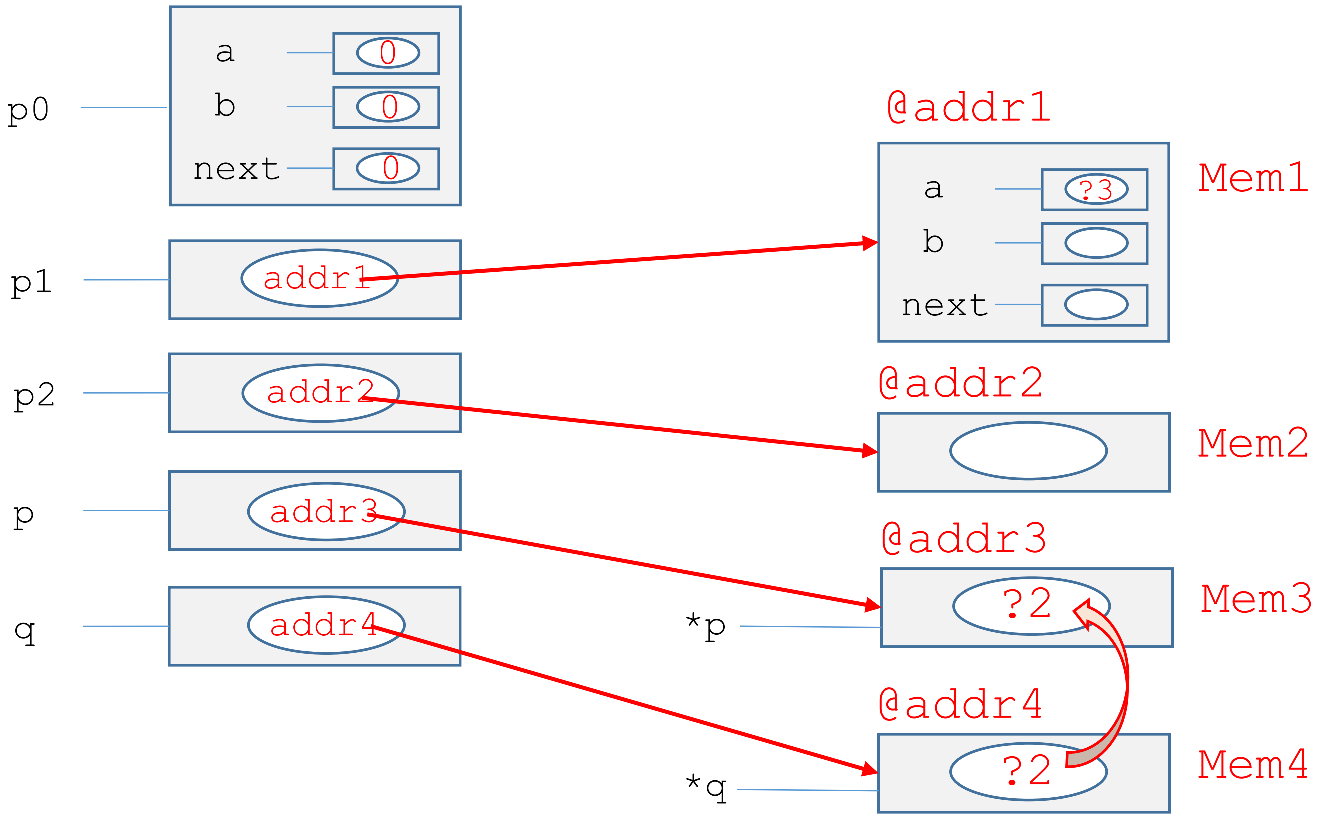
```
int main()
{
 p1 = (struct T *) malloc(sizeof(struct T));
 p2 = (struct T **) malloc(sizeof(struct T *));

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

 *p = *q;
```

main()



```
int main()
{
 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;
```

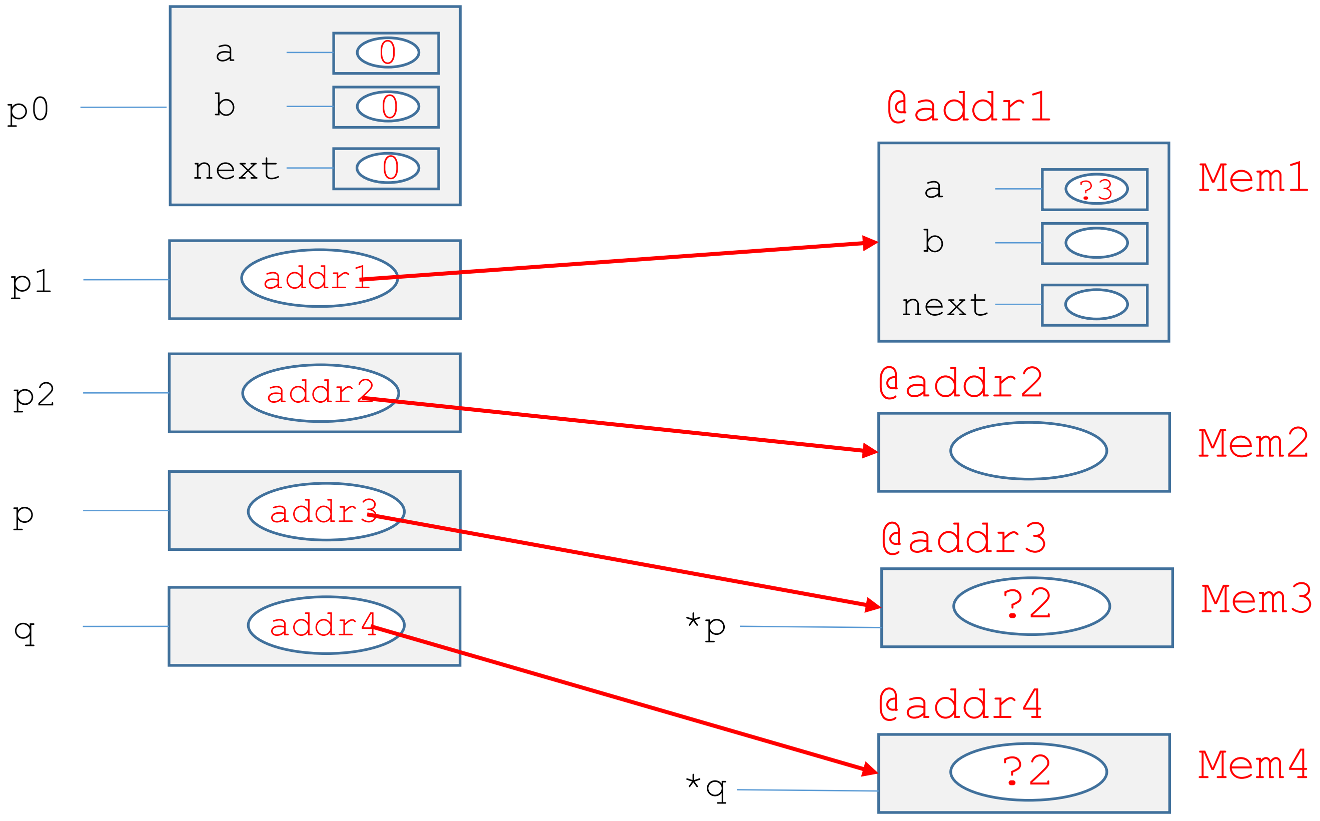
main()



The values in Mem3 and Mem4 are the same unknown value

main()

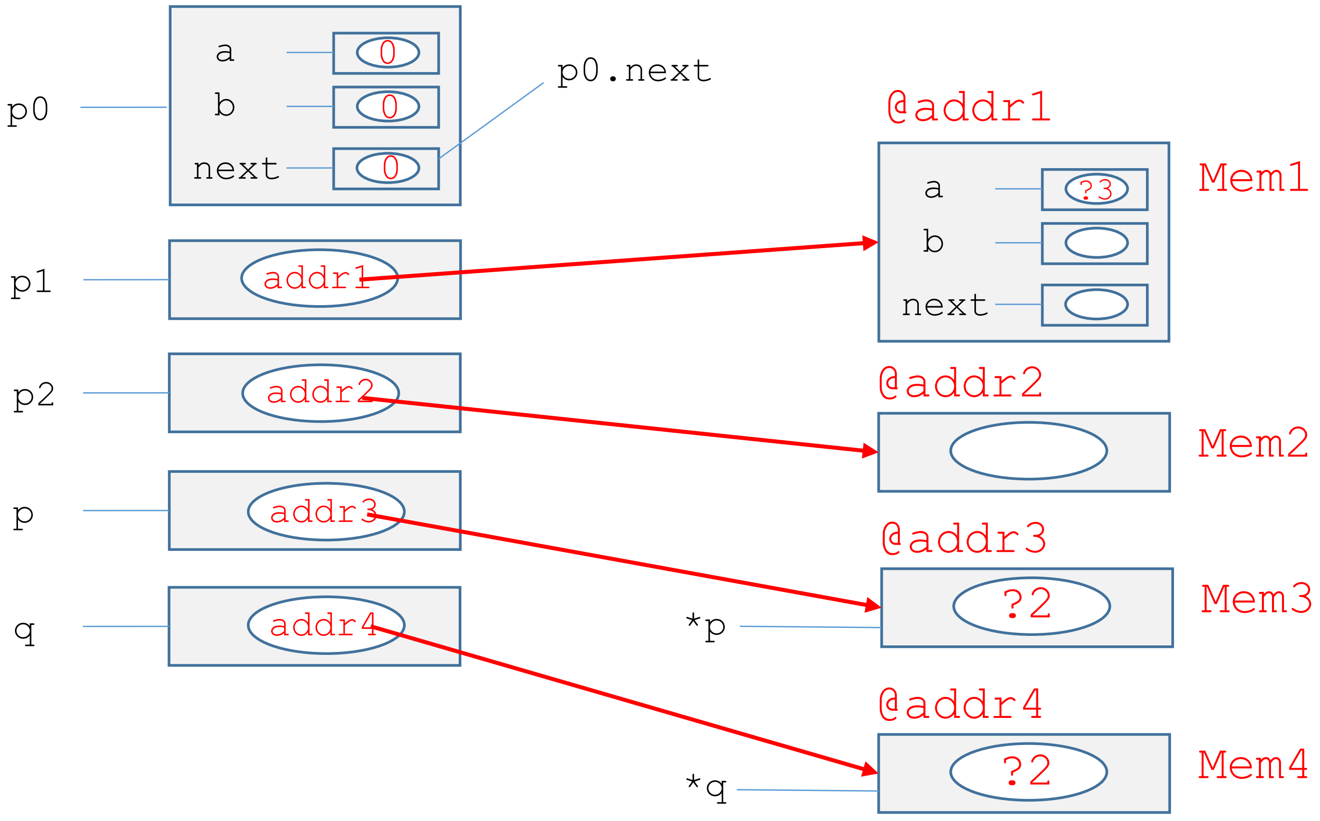
```
int main()
{
     ...

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



We continue with our program ...

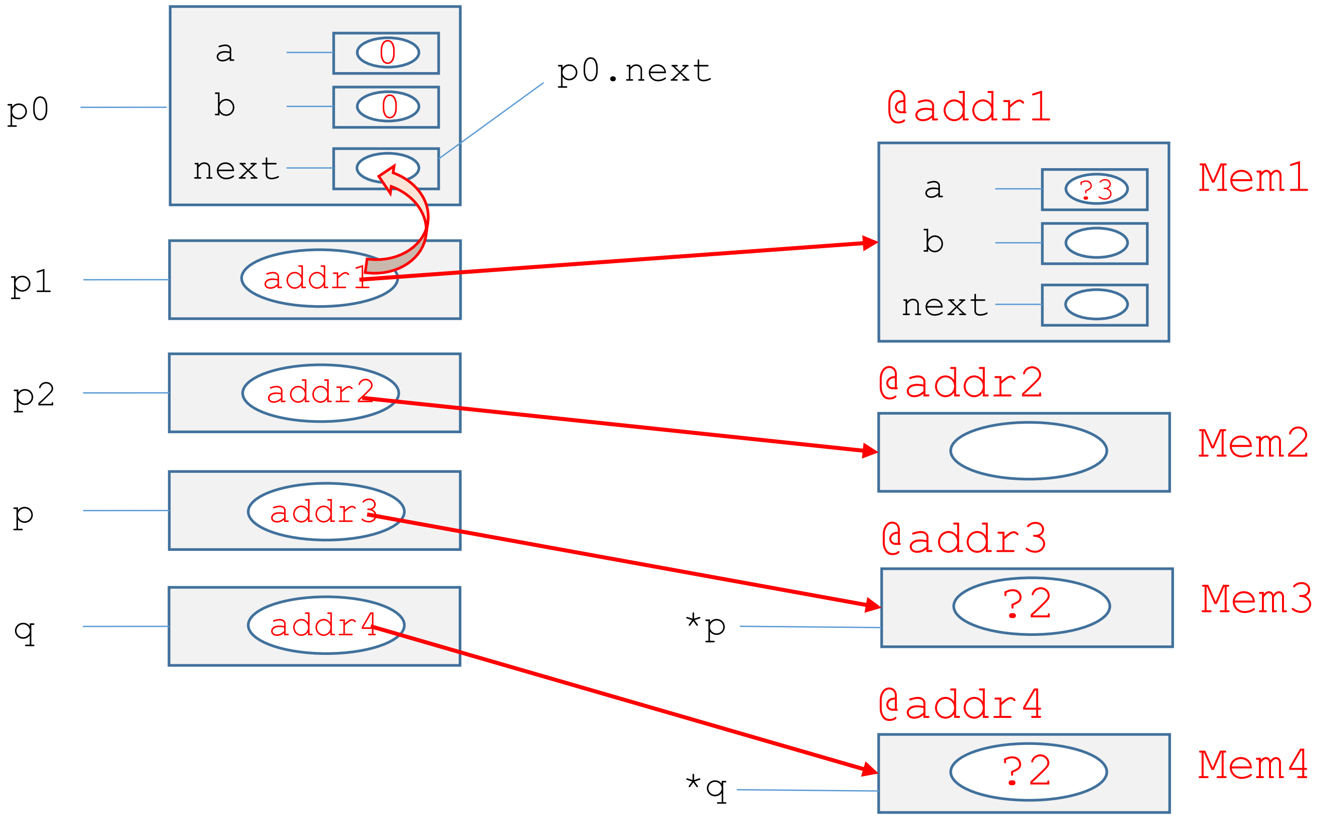
main()

```
int main()
{
     ...

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



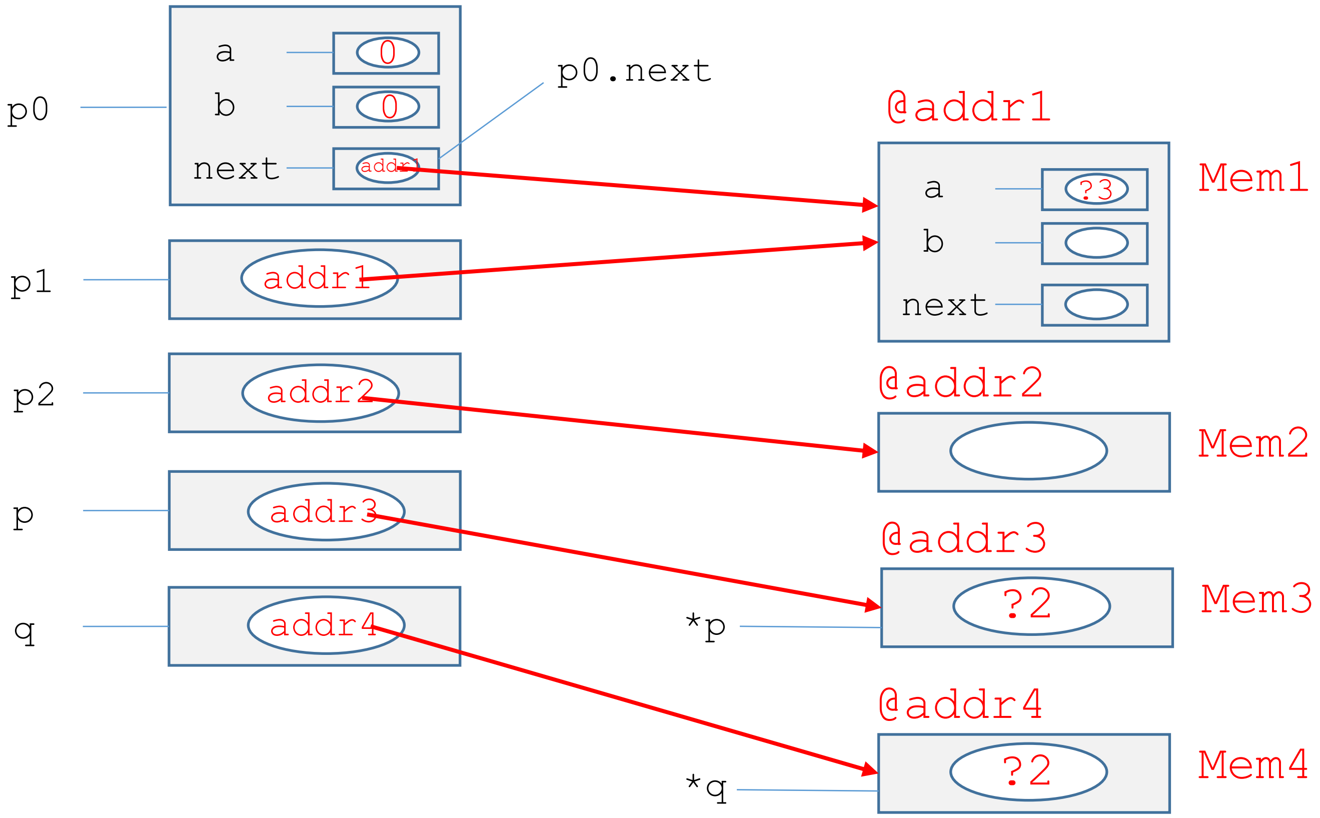
main()

```
int main()
{
     ...

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



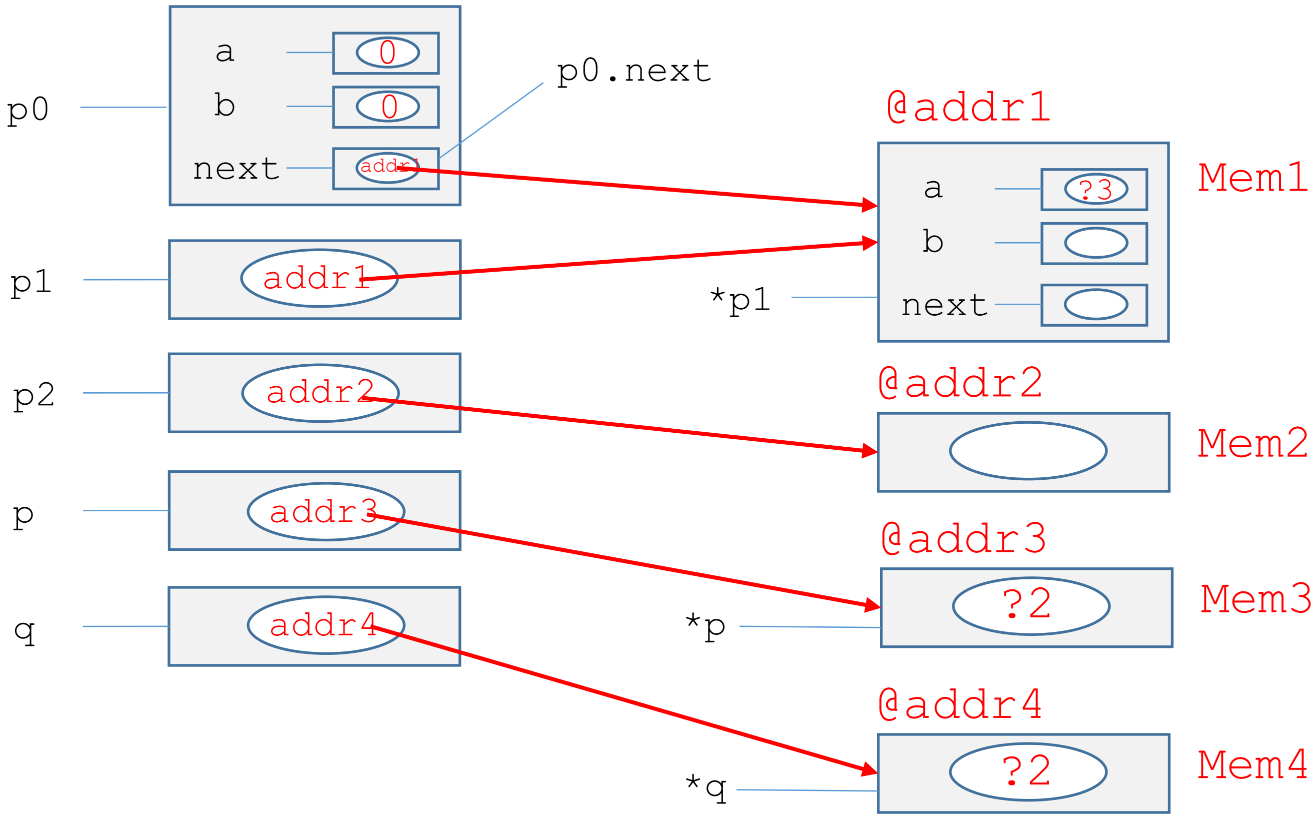
main()

```
int main()
{
     ...

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



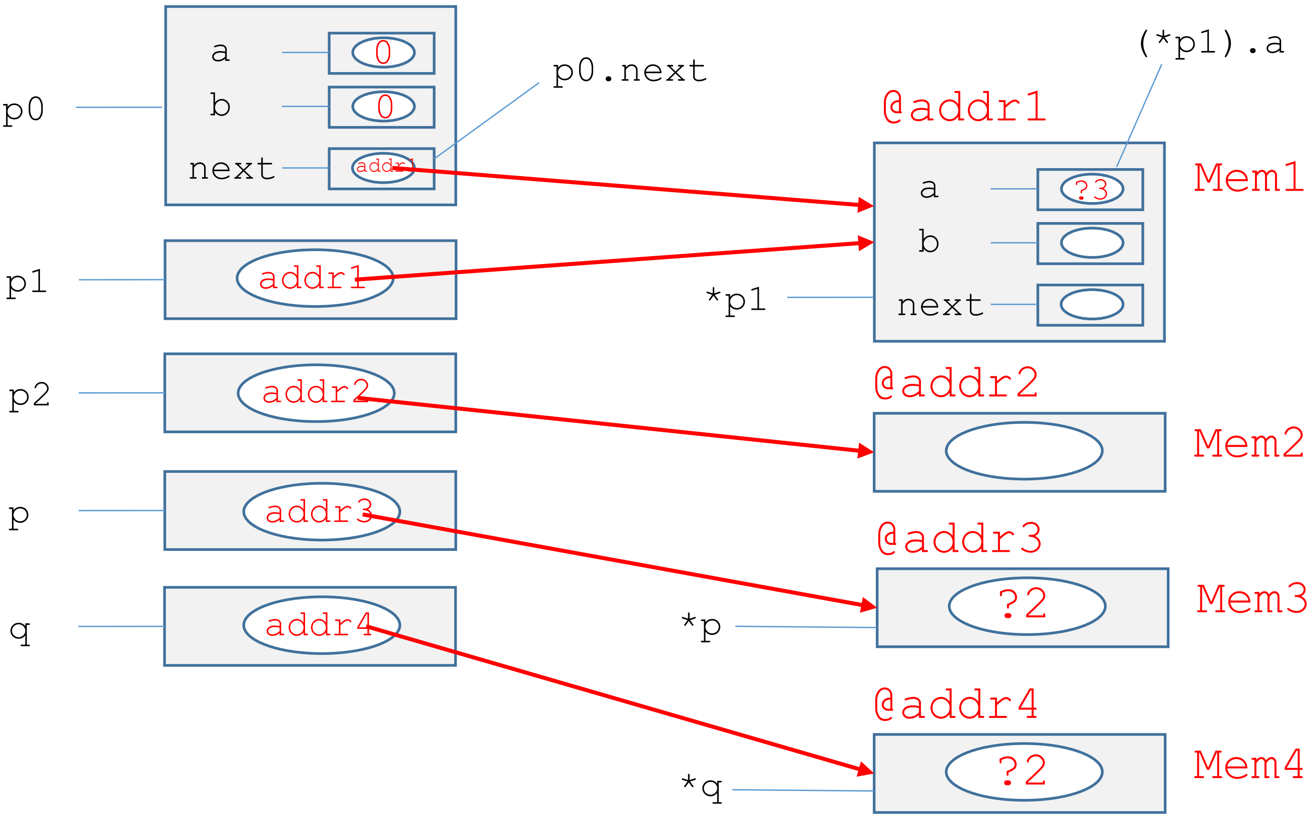
main()

```
int main()
{
     ...

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



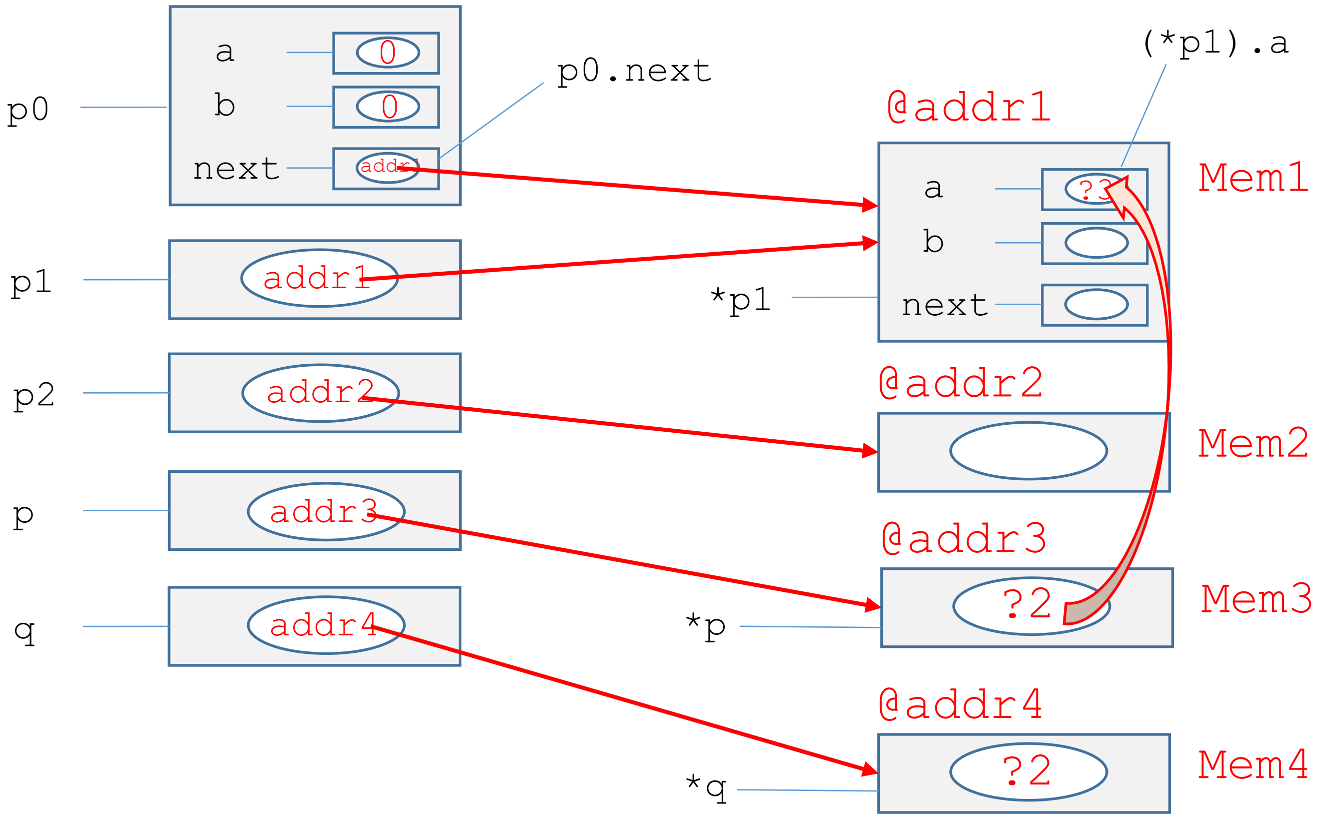
main()

```
int main()
{
    ...

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



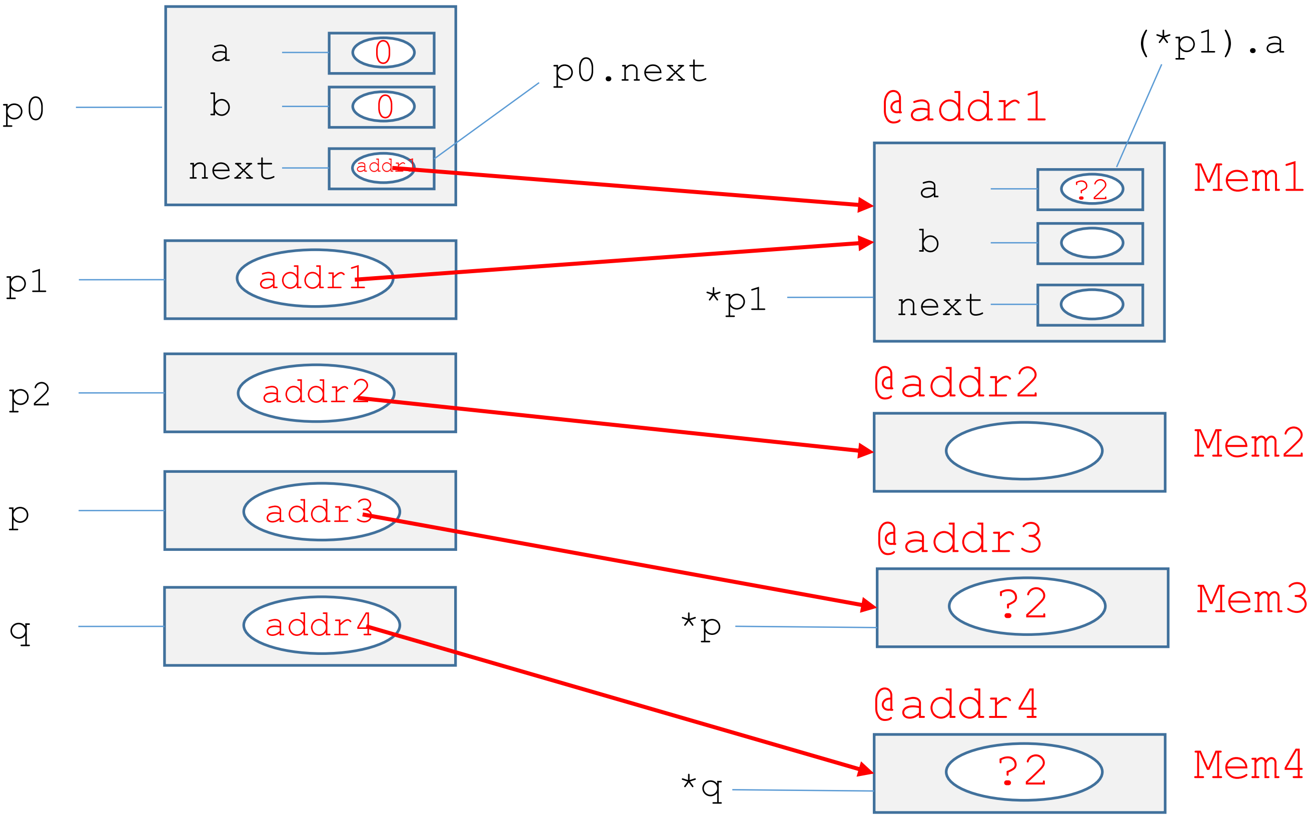
main()

```
int main()
{
    ...

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



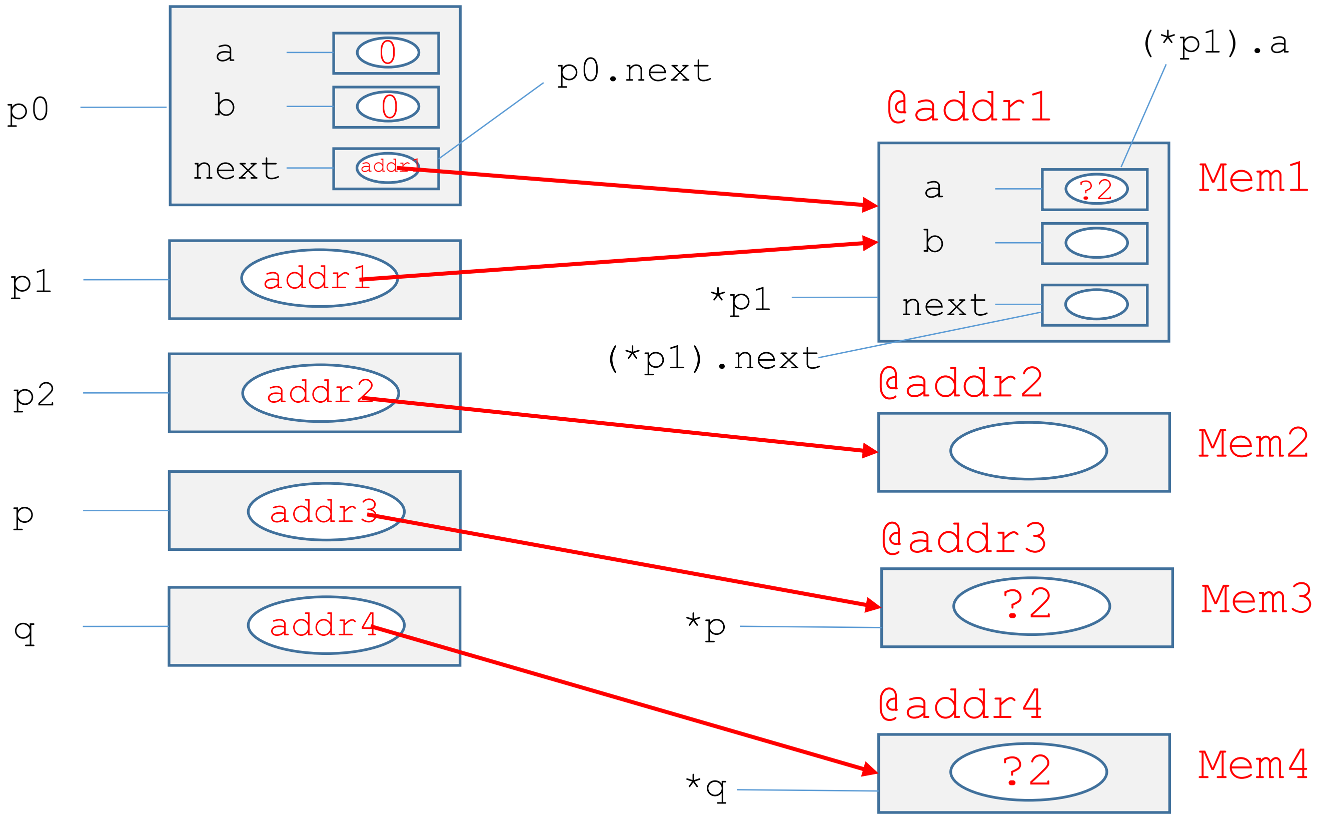
main()

```
int main()
{
     ...

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



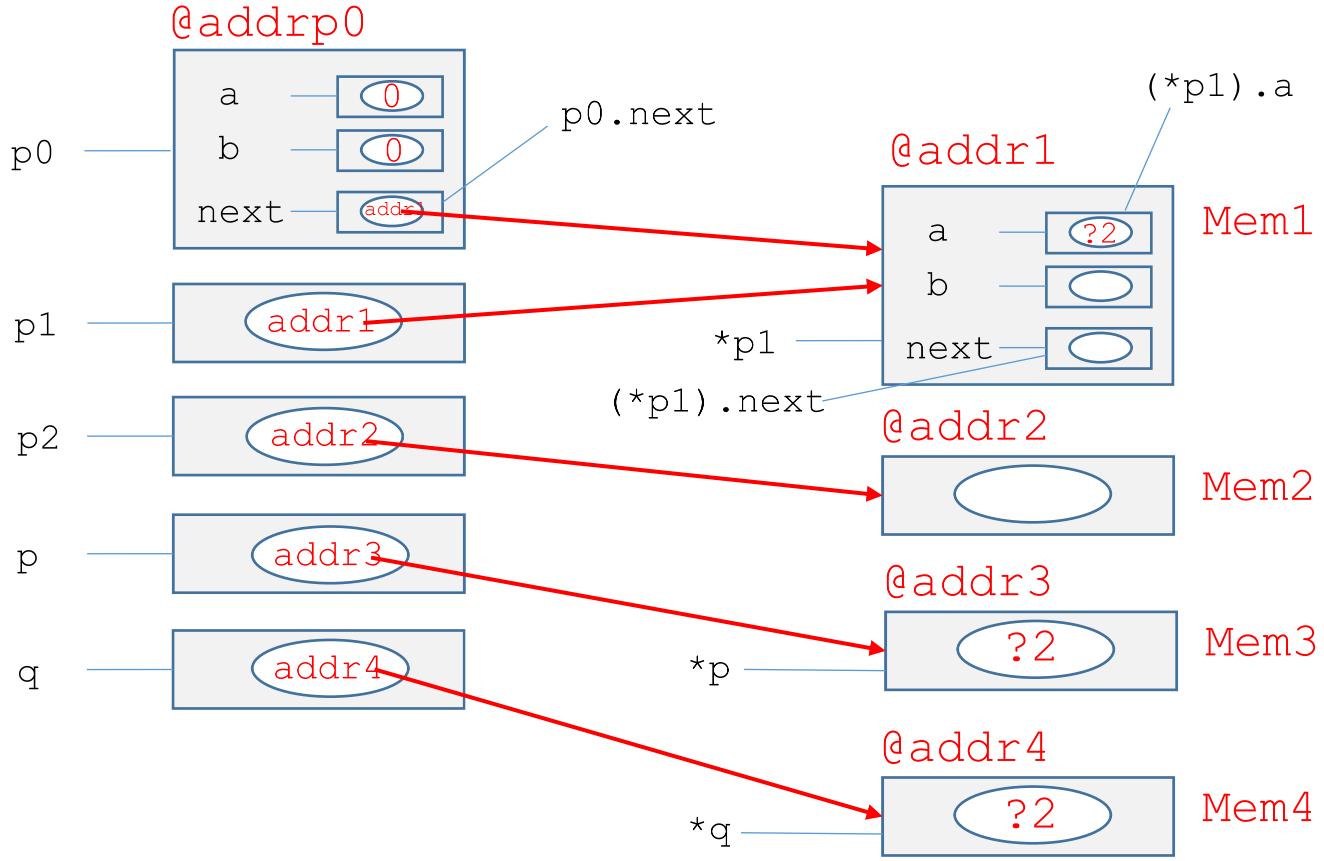
main()

```
int main()
{
     ...

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



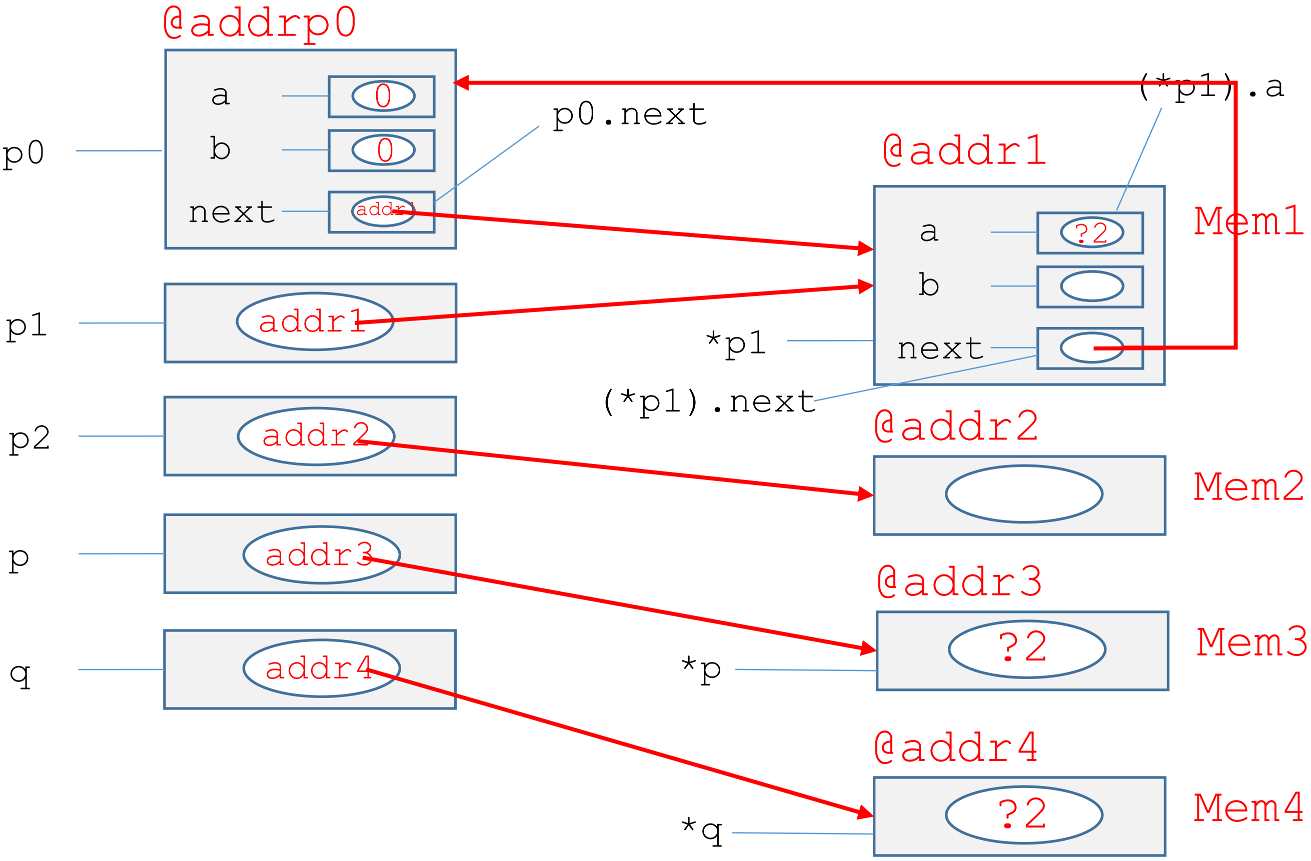
main()

```
int main()
{
     ...

  p0.next = p1;
  (*p1).a = *p;
  (*p1).next  = &p0;    //&p0 = @addrp0
  *p2 = (*p1).next;
```



```
int main()
{
    ...

  p0.next = p1;
  (*p1).a = *p;
  (*p1).next = &p0;    //&p0 = @addrp0
  *p2 = (*p1).next;
```

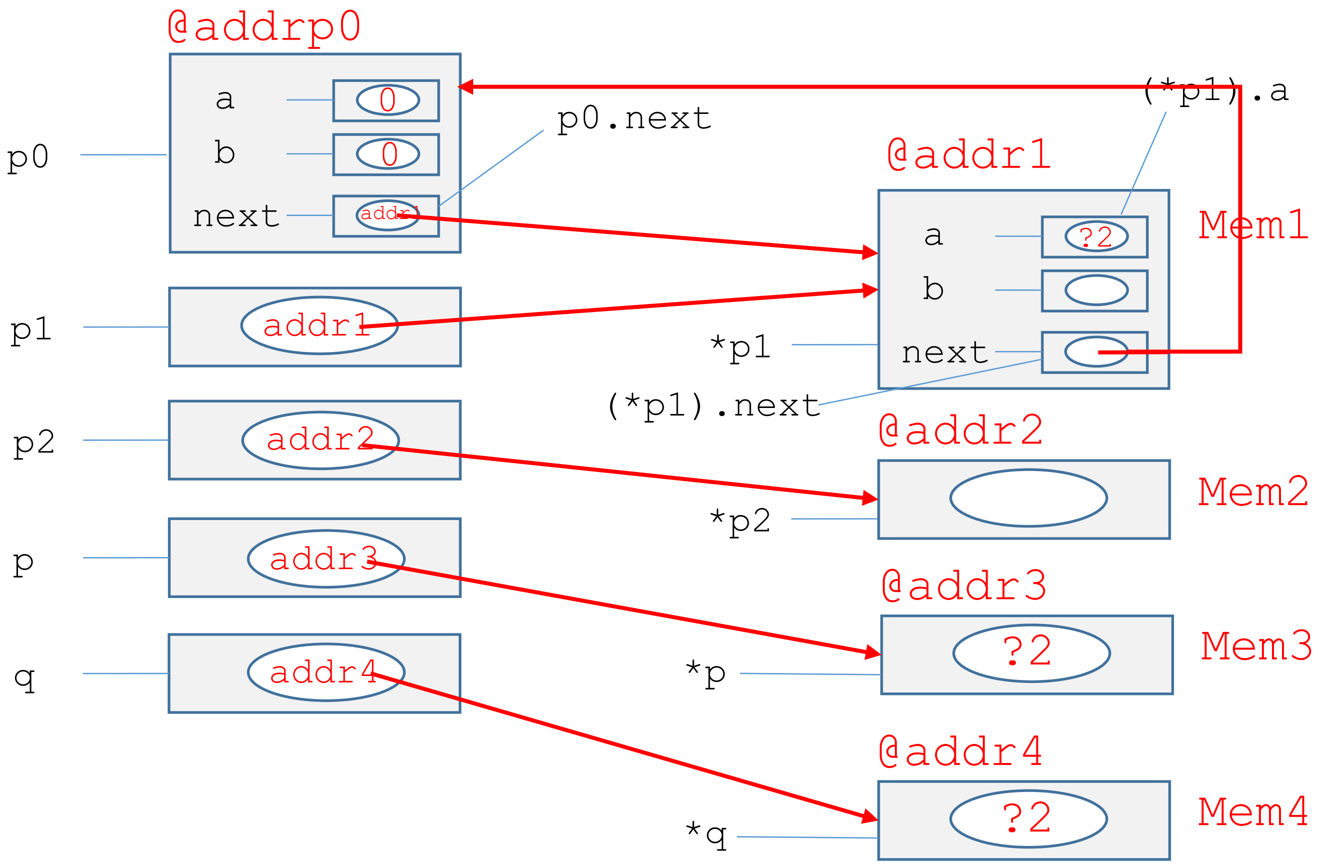
main()



(*p1).next is the same as p1->next
p1->next is syntactic sugar for (*p1).next

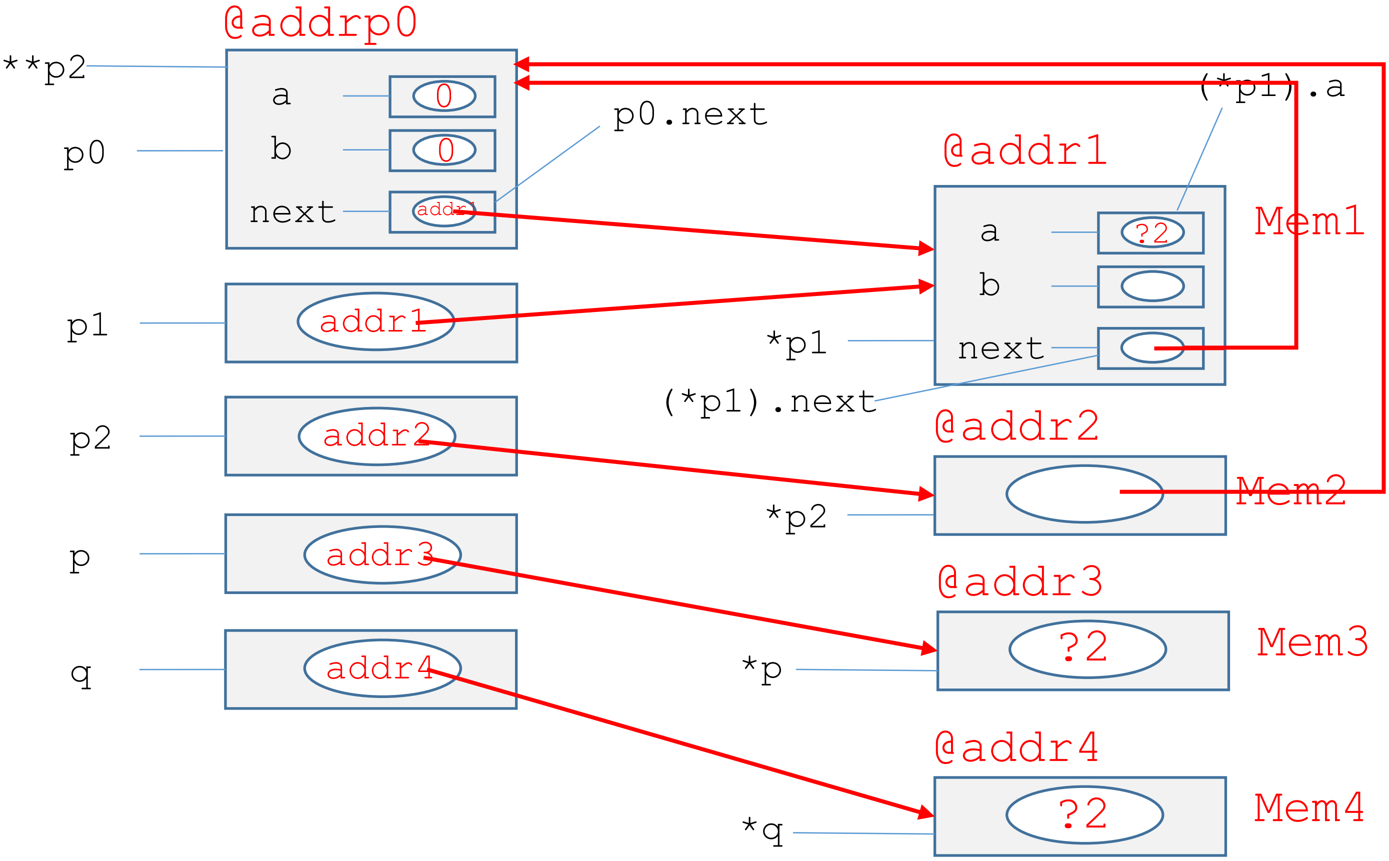
main()

```
int main()
{
     ...

  p0.next = p1;
  (*p1).a = *p;
  (*p1).next = &p0;    //&p0 = @addrp0
  *p2 = (*p1).next;
```



main()

```
int main()
{
    ...

  p0.next = p1;
  (*p1).a = *p;
  (*p1).next = &p0;    //&p0 = @addrp0
  *p2 = (*p1).next;
```



&p and p are not the same

&p is an r-value p is an l-value

&p is just a number p is not a number.

It is the name of a location

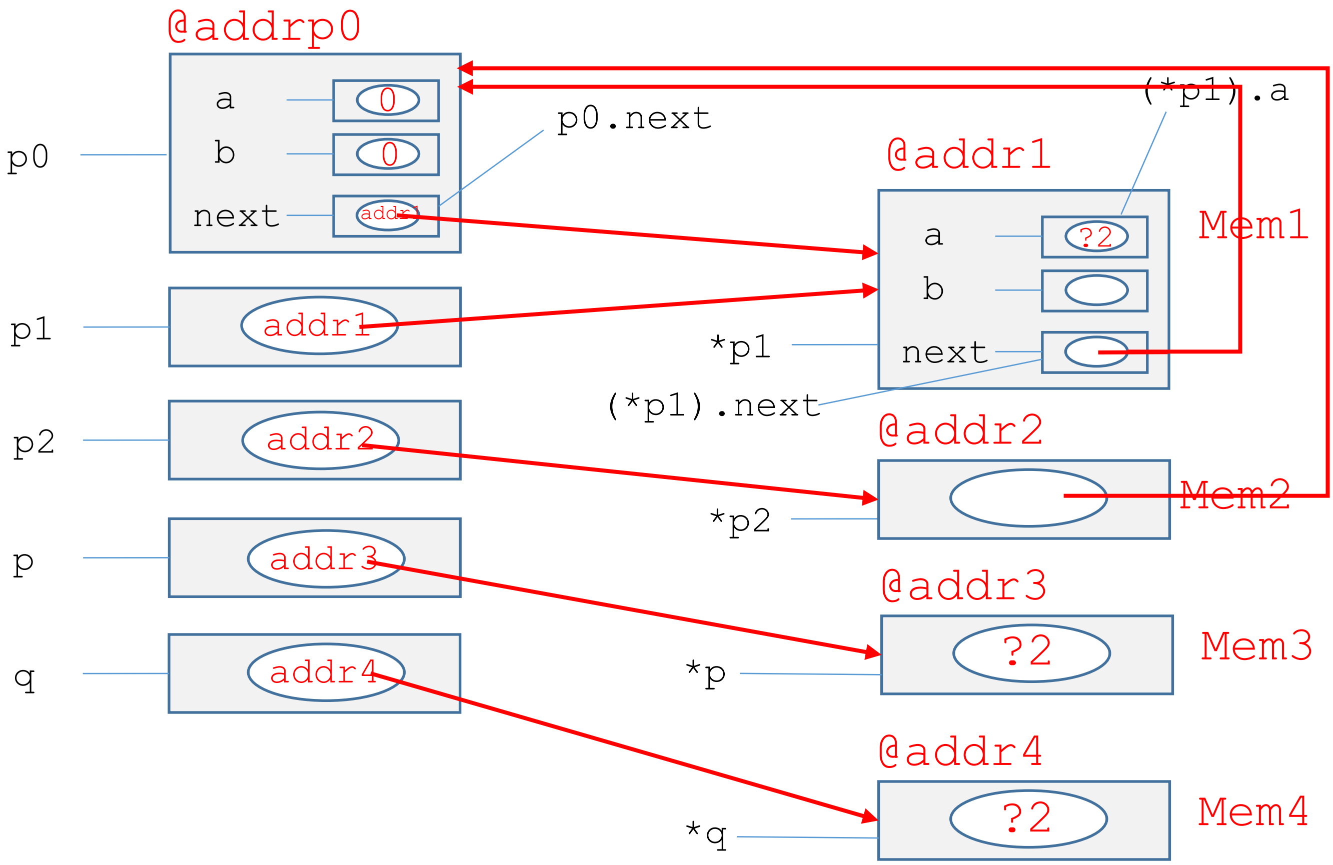
*&p is an alias of p, but *&p is not the same as &p

*&p = … ok

&p = … not ok

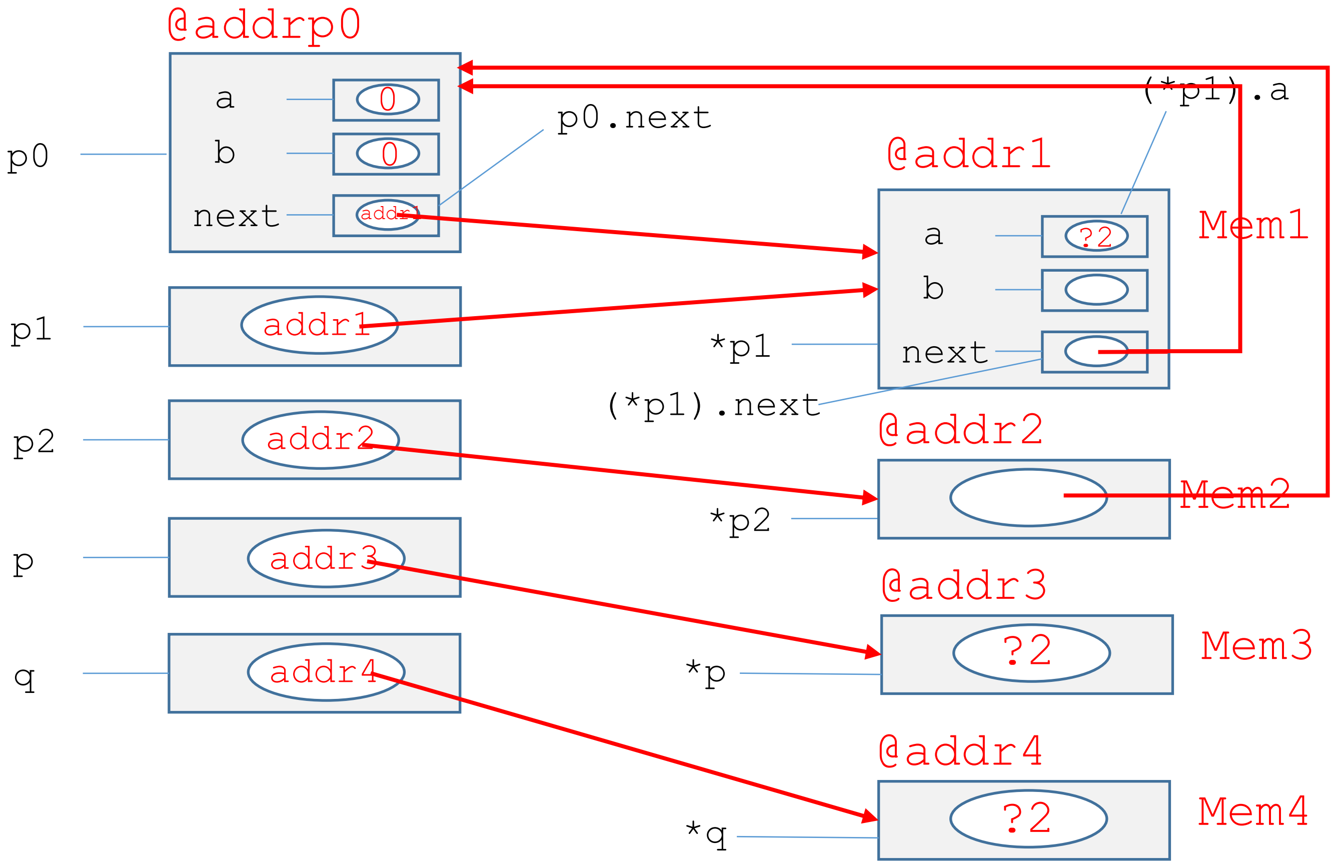
main()

```
int main()
{
     ...

  p0.next = p1;
  (*p1).a = *p;
  (*p1).next = &p0;    //&p0 = @addrp0
  *p2 = (*p1).next;
```



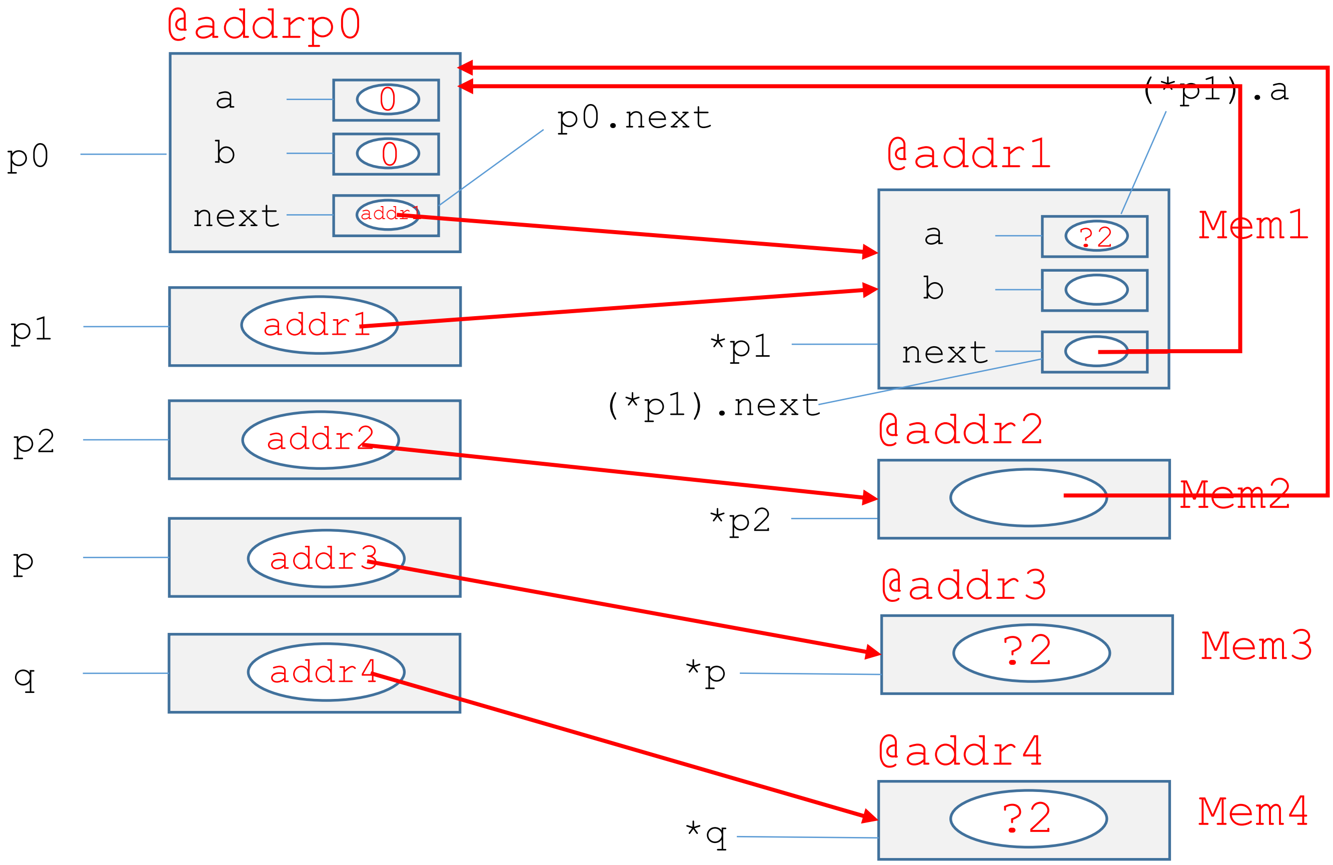
We continue with our program ...

```
int main()
{
     ...
     {  // NEW SCOPE
       struct T *a[5];

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

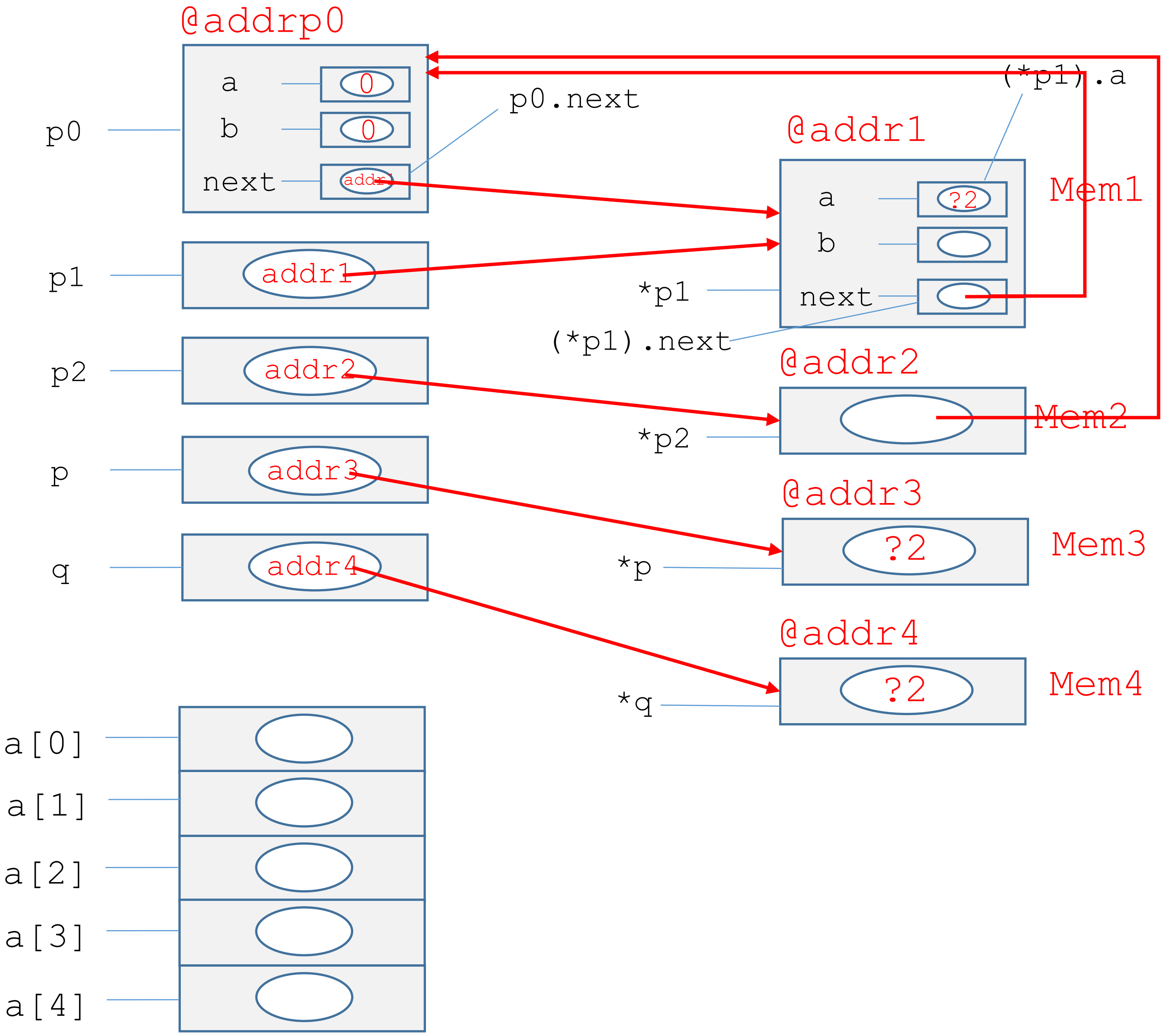


We continue with our program ...
with new declaration and statements

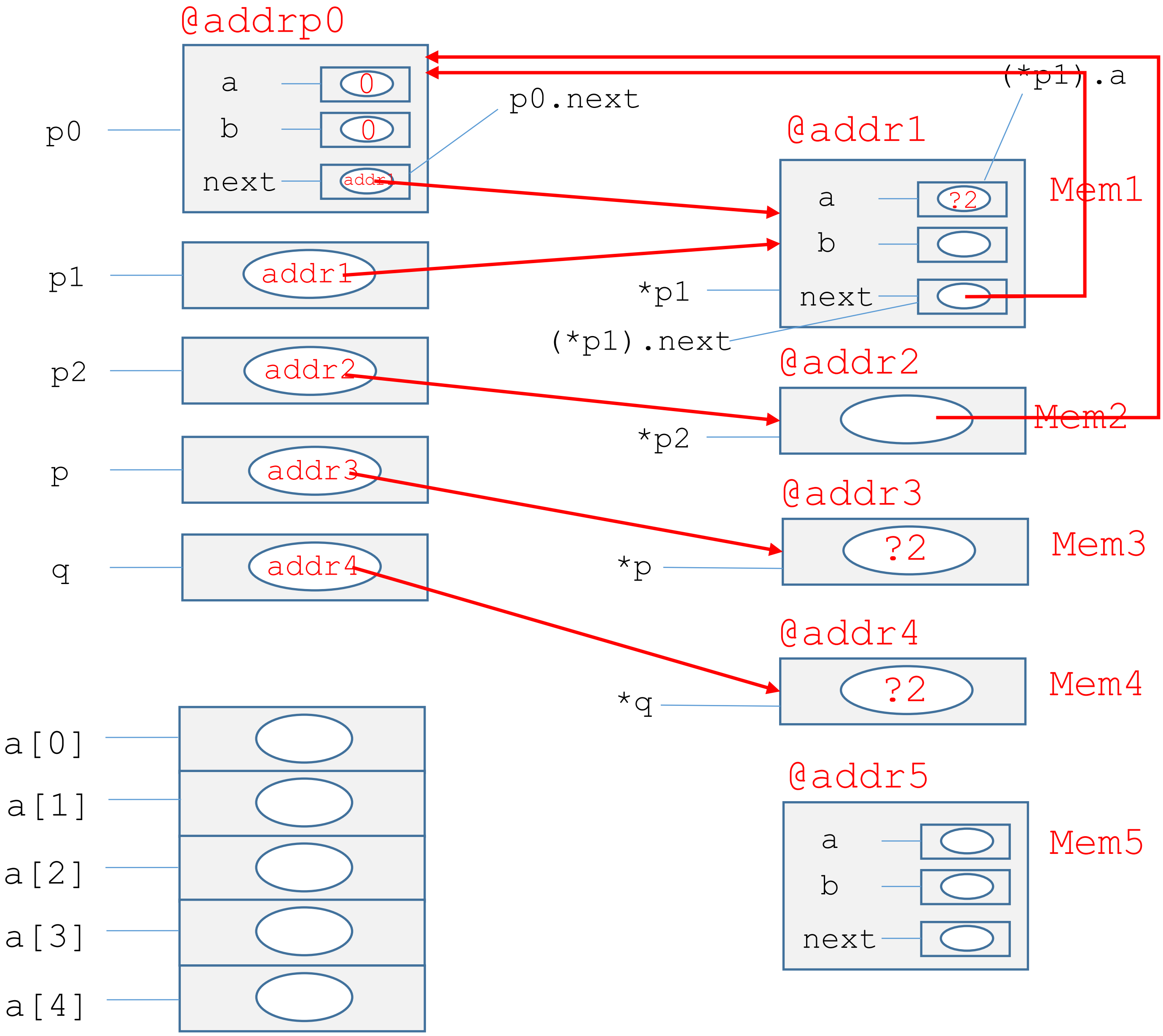
```
int main()
{
      ...
      {  // NEW SCOPE
        struct T *a[5];

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



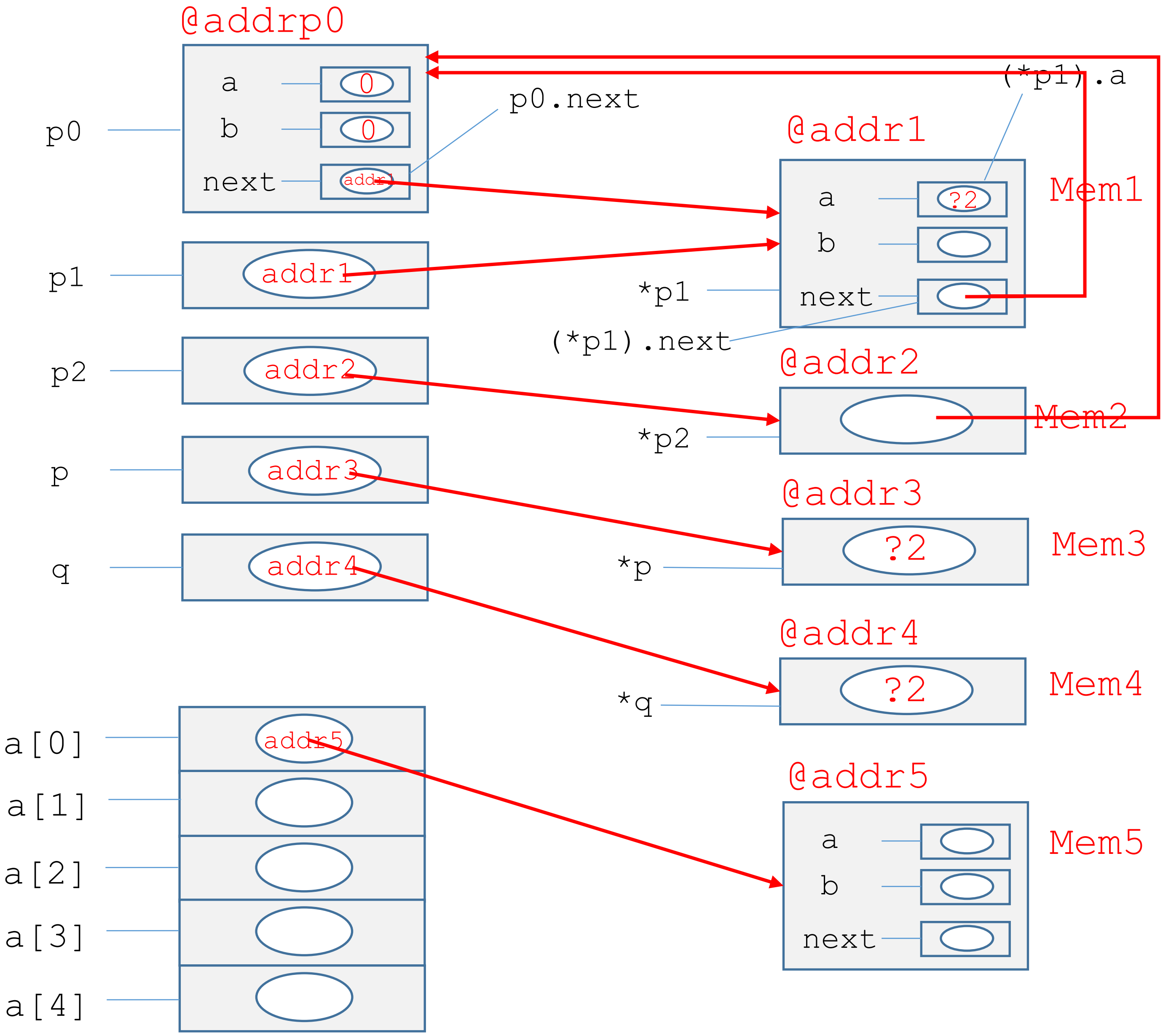
```
int main()
{
     ...
     {
        struct T *a[5];

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



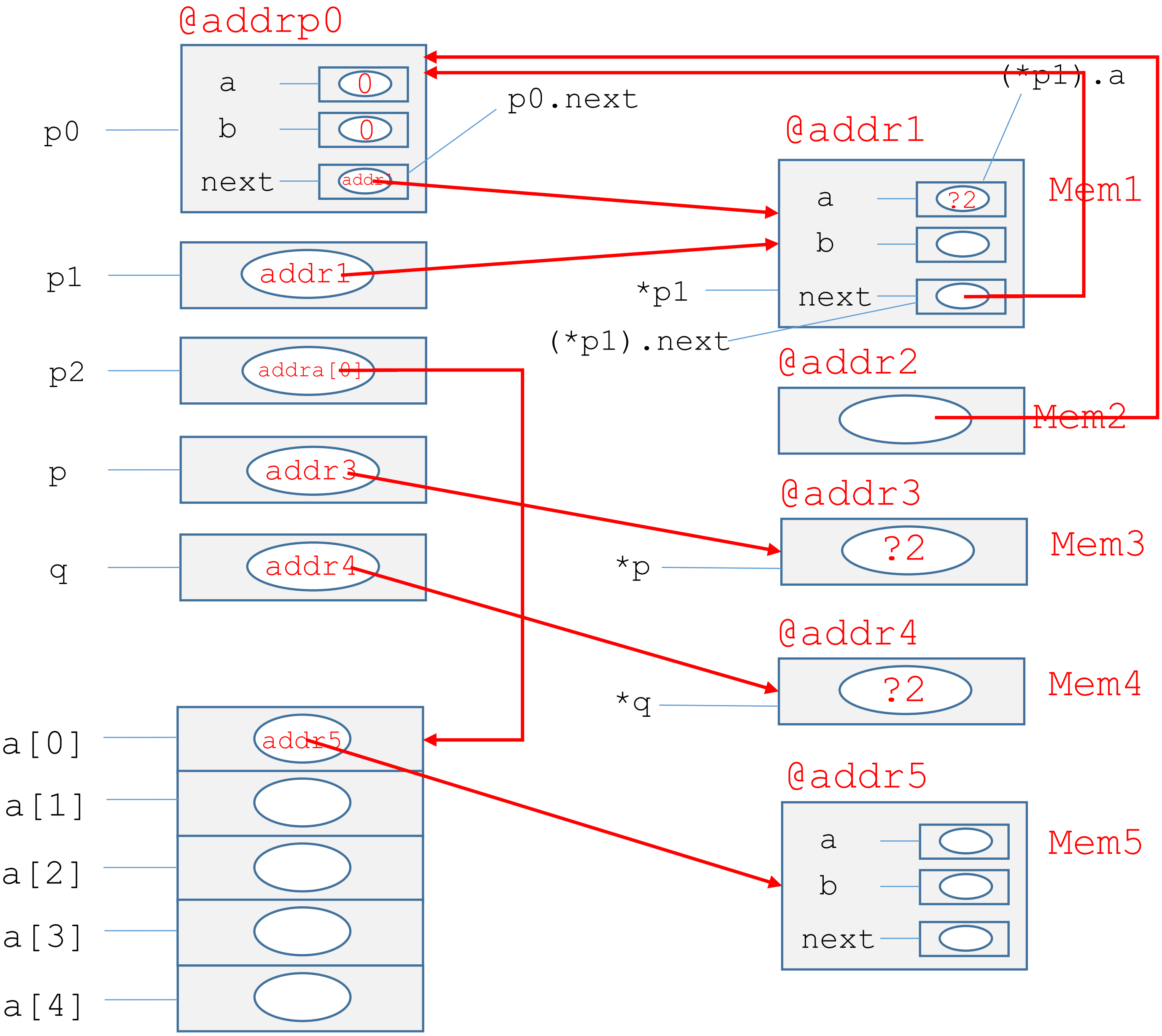
```
int main()
{
     ...
     {
       struct T *a[5];

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



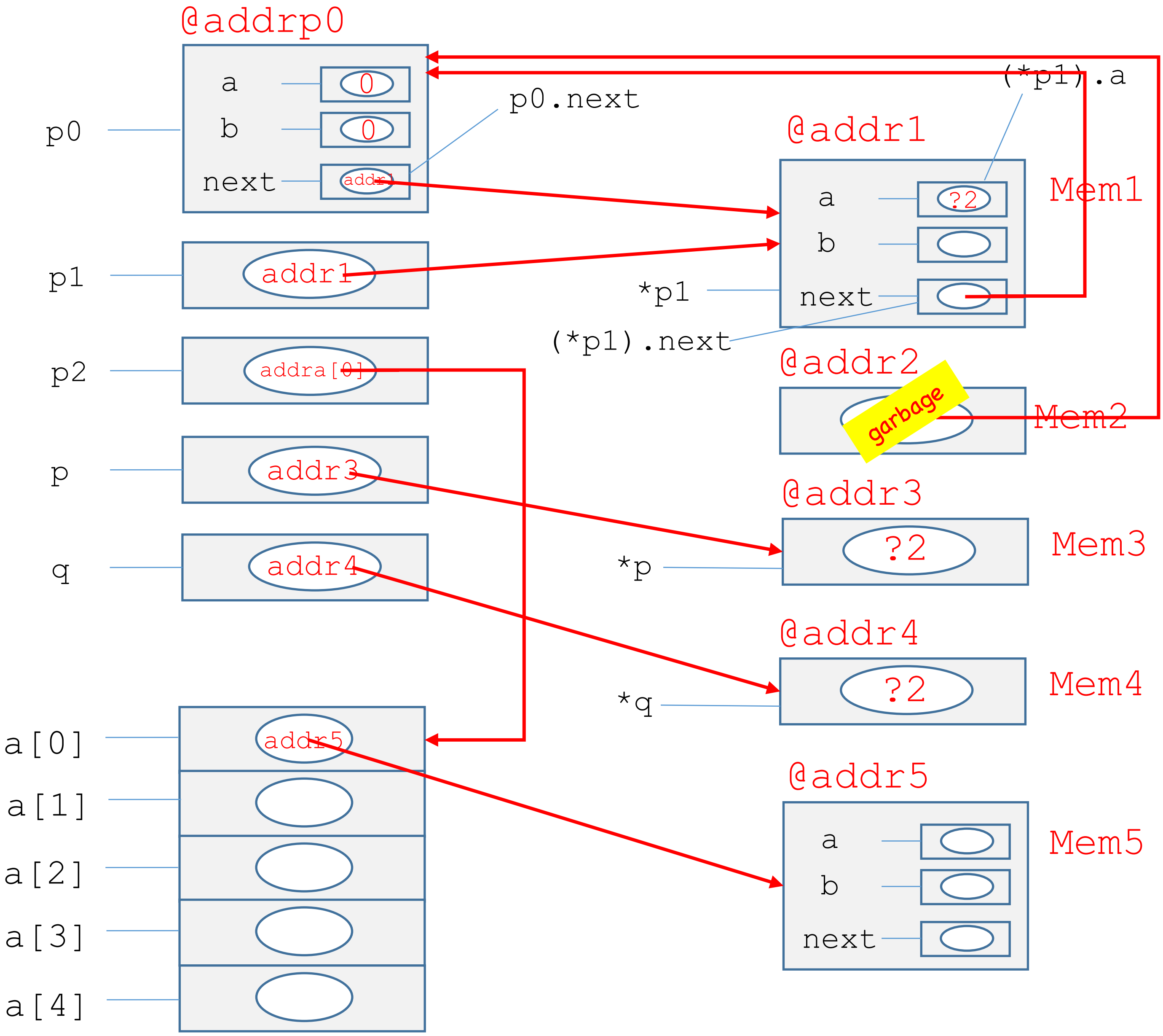
```
int main()
{
     ...
     {
       struct T *a[5];

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



```
int main()
{
     ...
     {
       struct T *a[5];

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

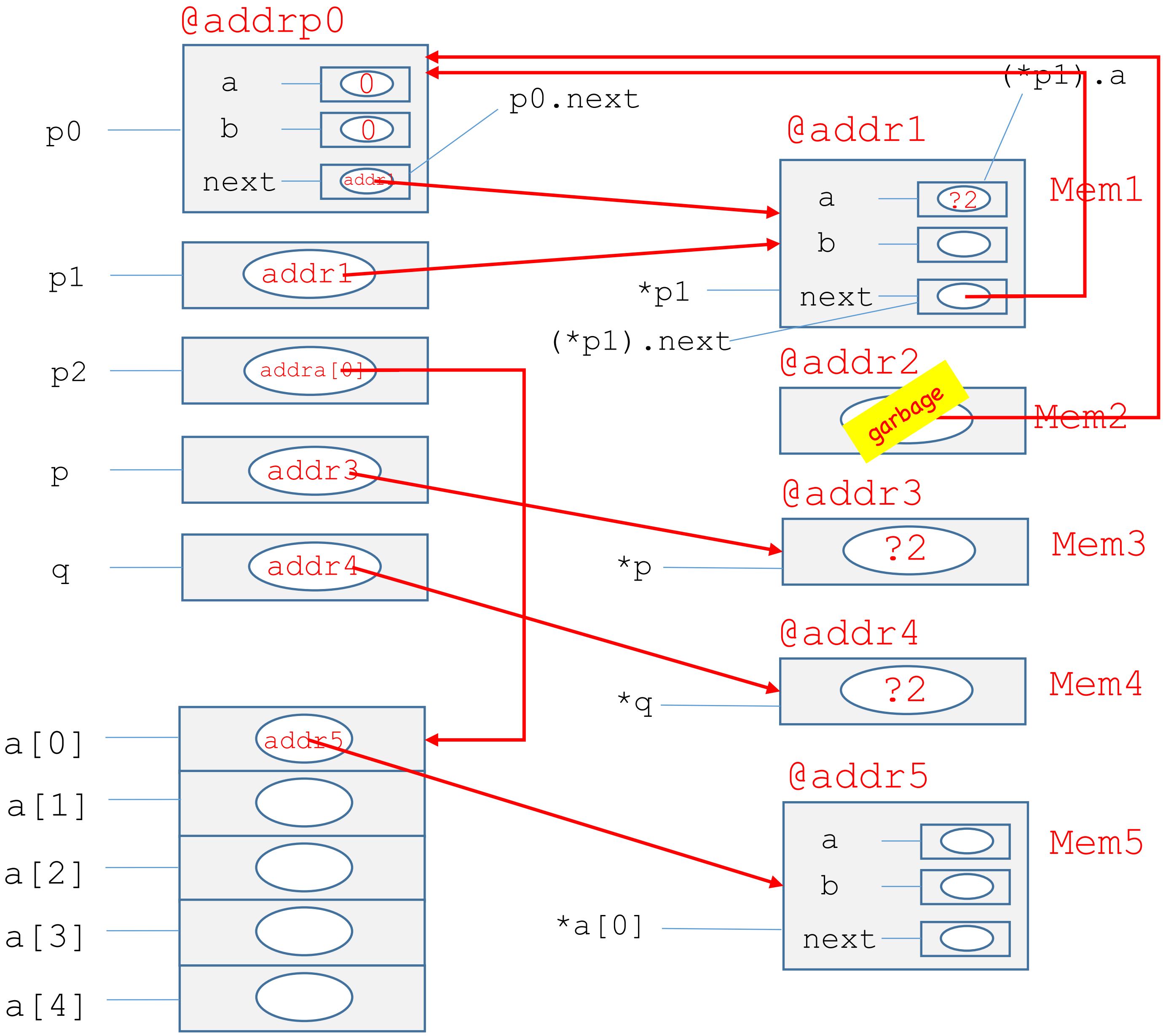


```
int main()
{
     ...
     {
       struct T *a[5];

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

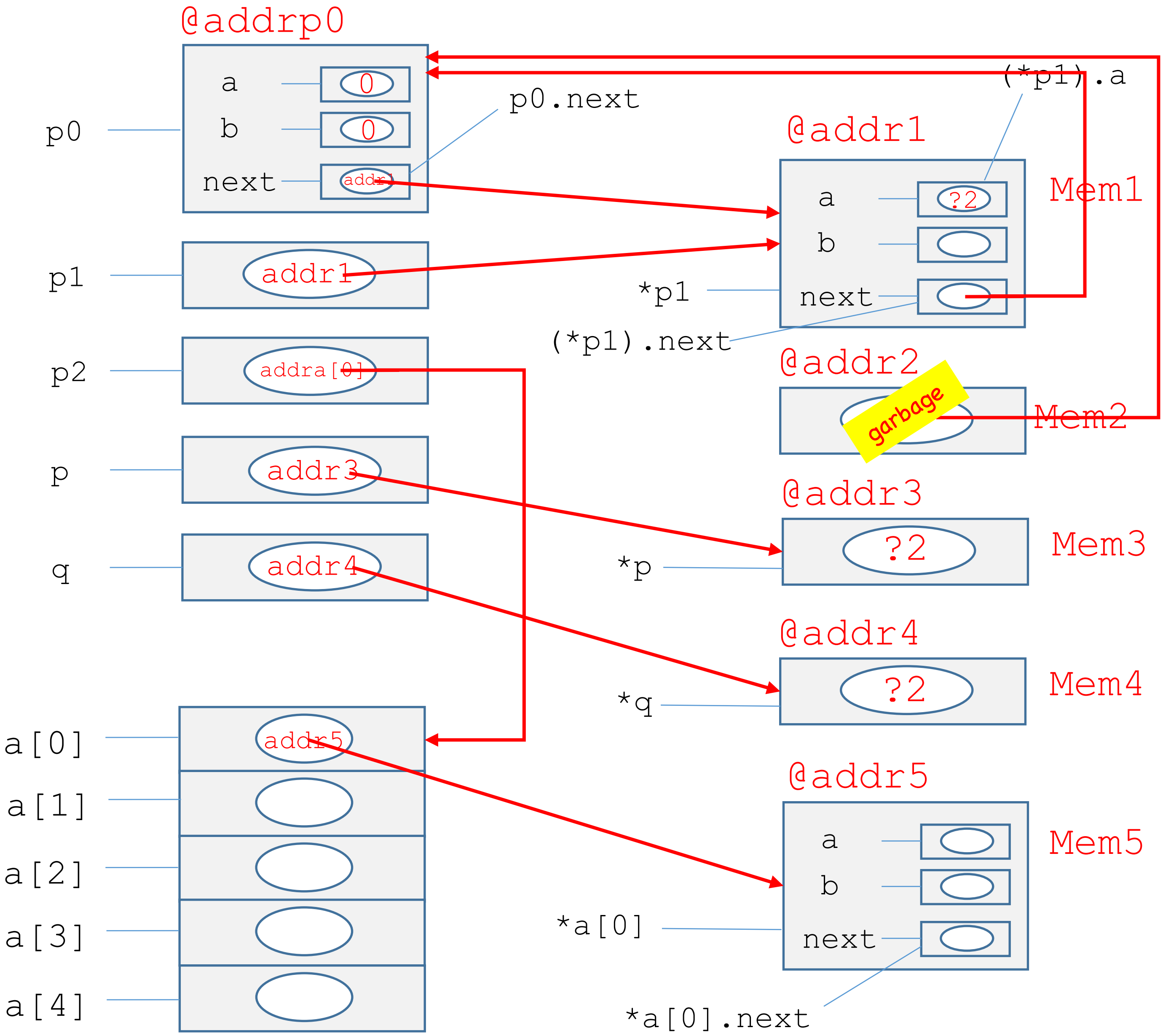


Mem2 is no longer accessible

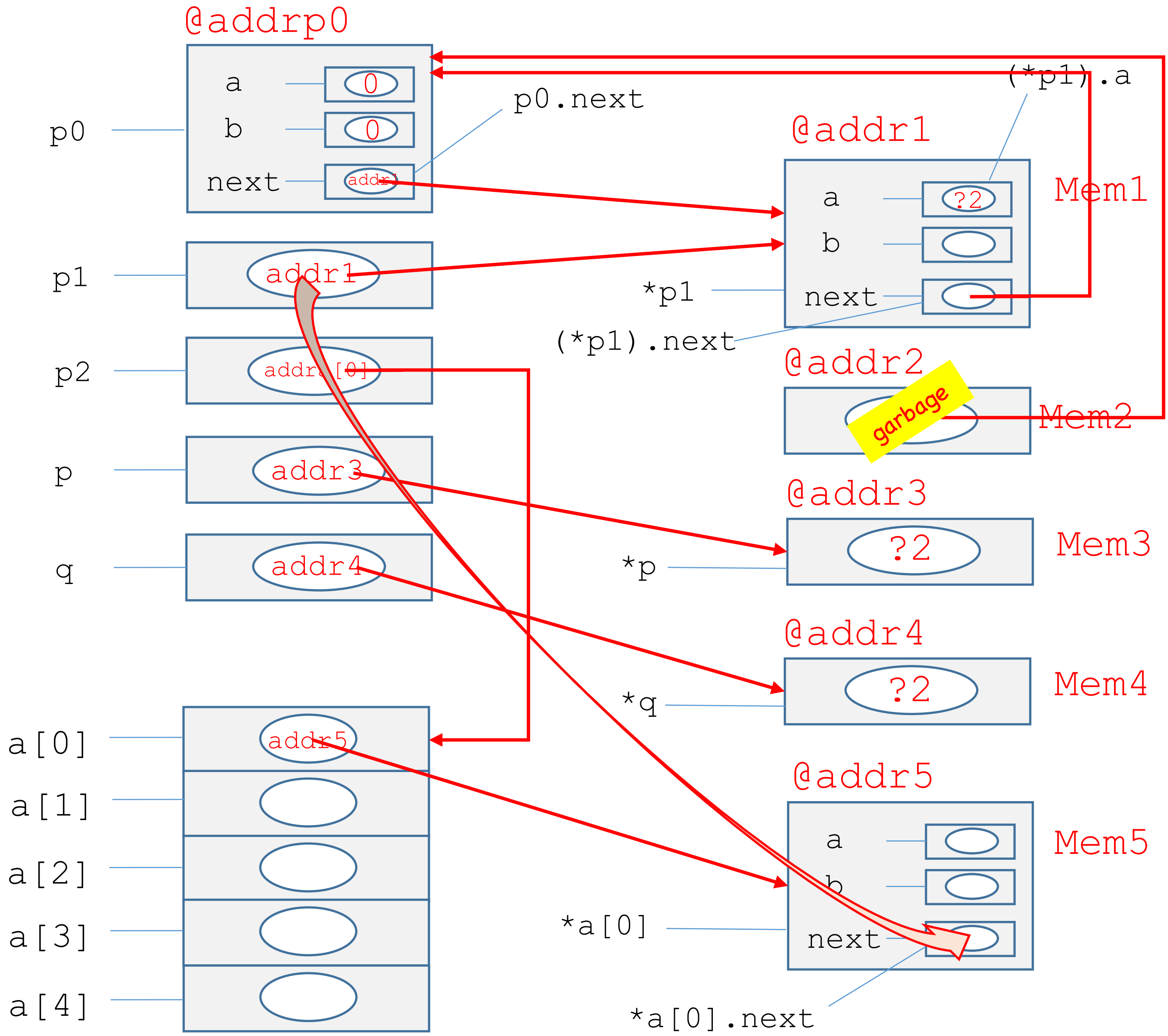
```
int main()
{
     ...
     {
       struct T *a[5];

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



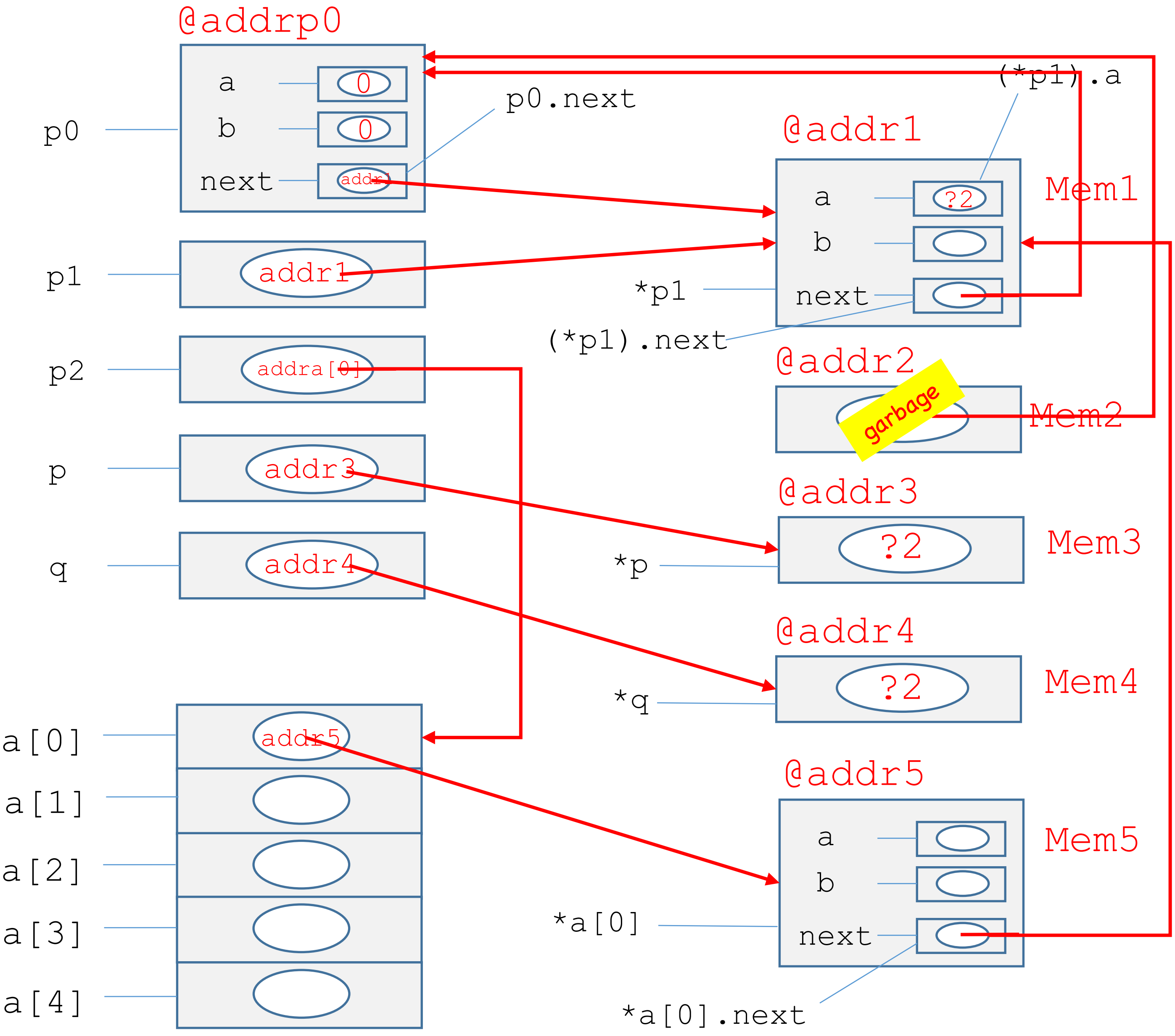
```
int main()
{
     ...
     {
       struct T *a[5];

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



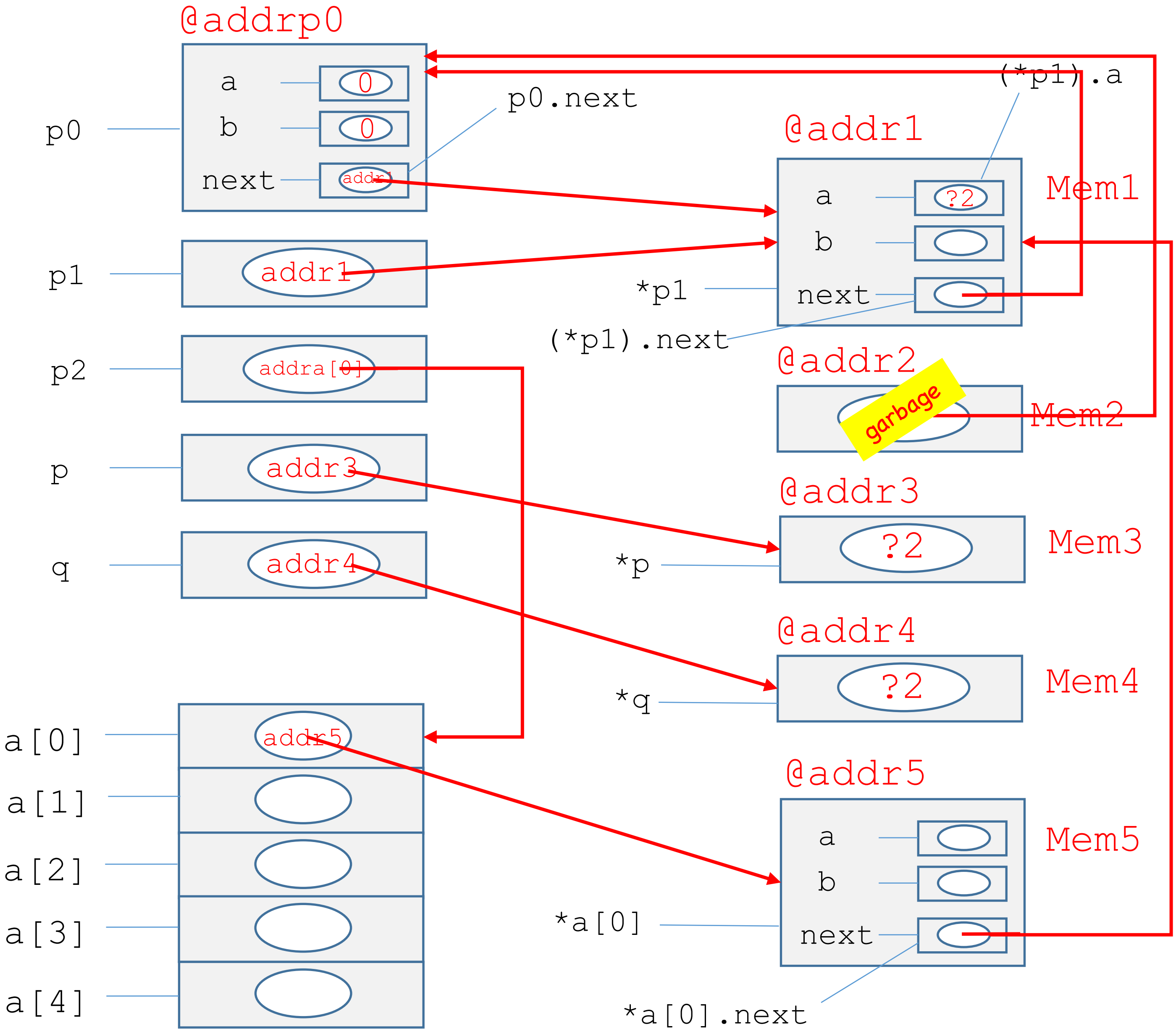
```
int main()
{
     ...
     {
       struct T *a[5];

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



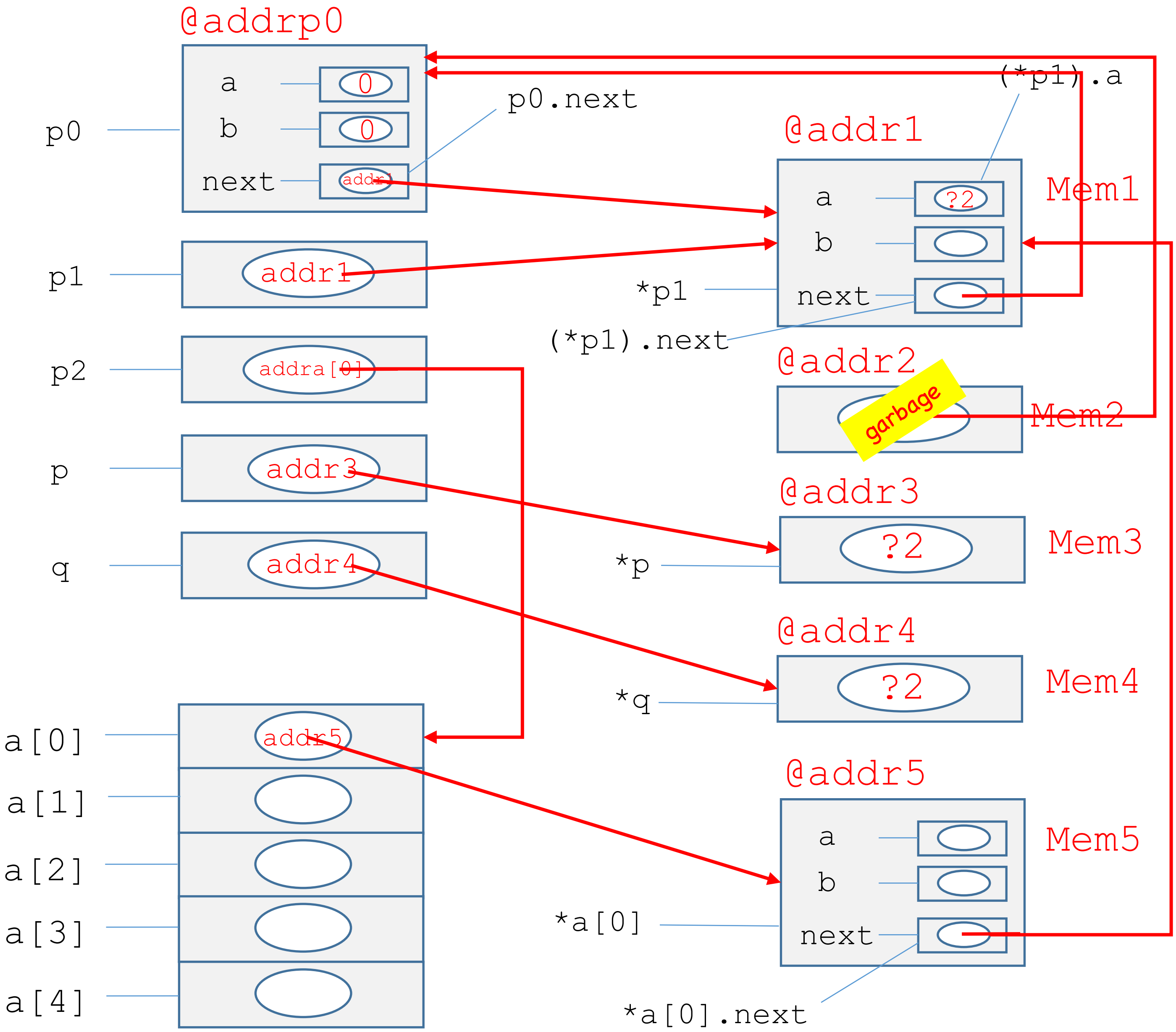
```
int main()
{
     ...
     {
       struct T *a[5];

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



```
int main()
{
      ...
      {
         struct T *a[5];

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

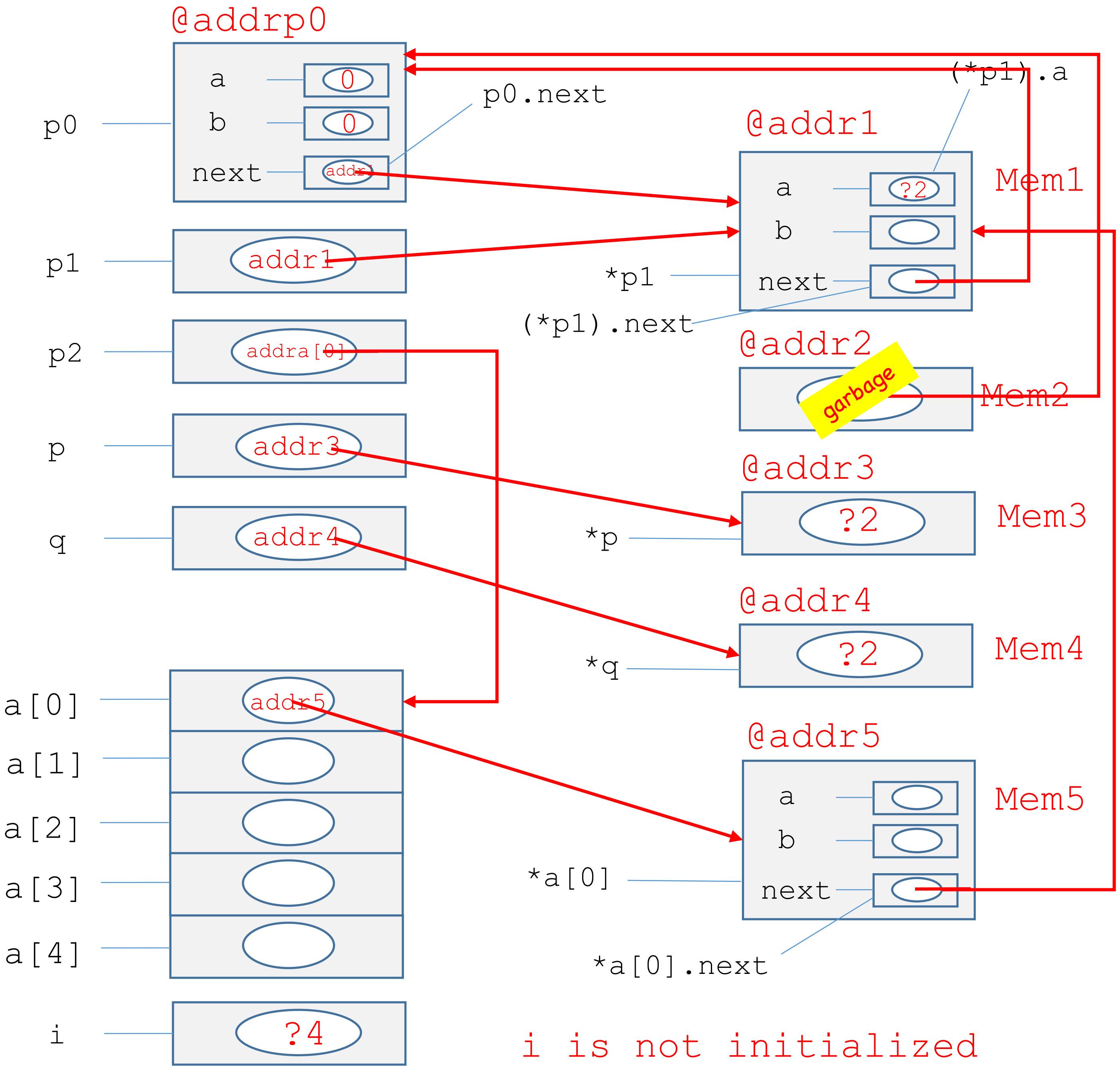


```
int main() {
     ...
     {
       struct T *a[5];
       ...
       int i;
       struct T* cursor;
```



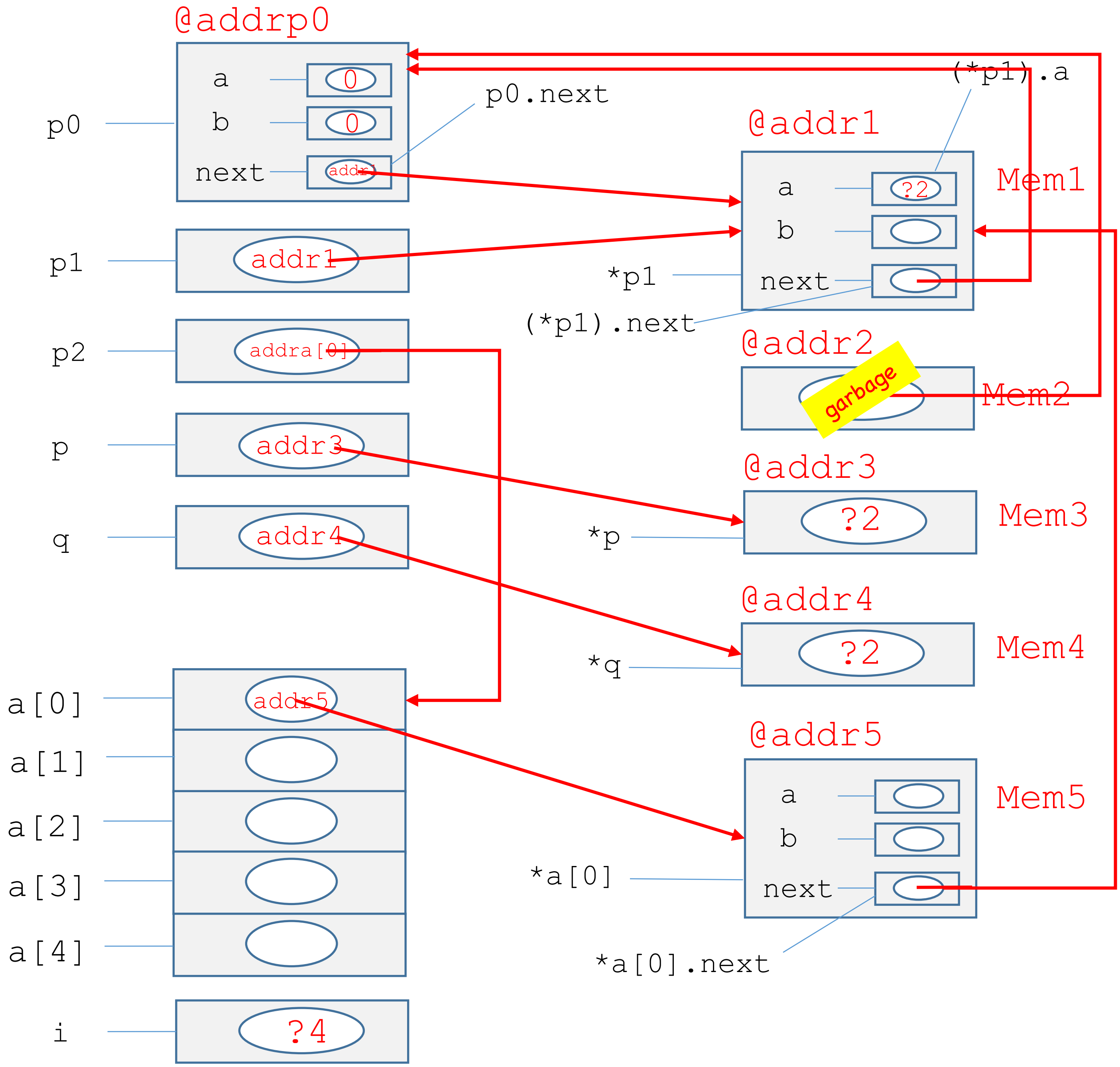
We continue with our program ...
with new declarations

```
int main() {
     ...
     {
       struct T *a[5];
       ...
       int i;
       struct T* cursor;
```

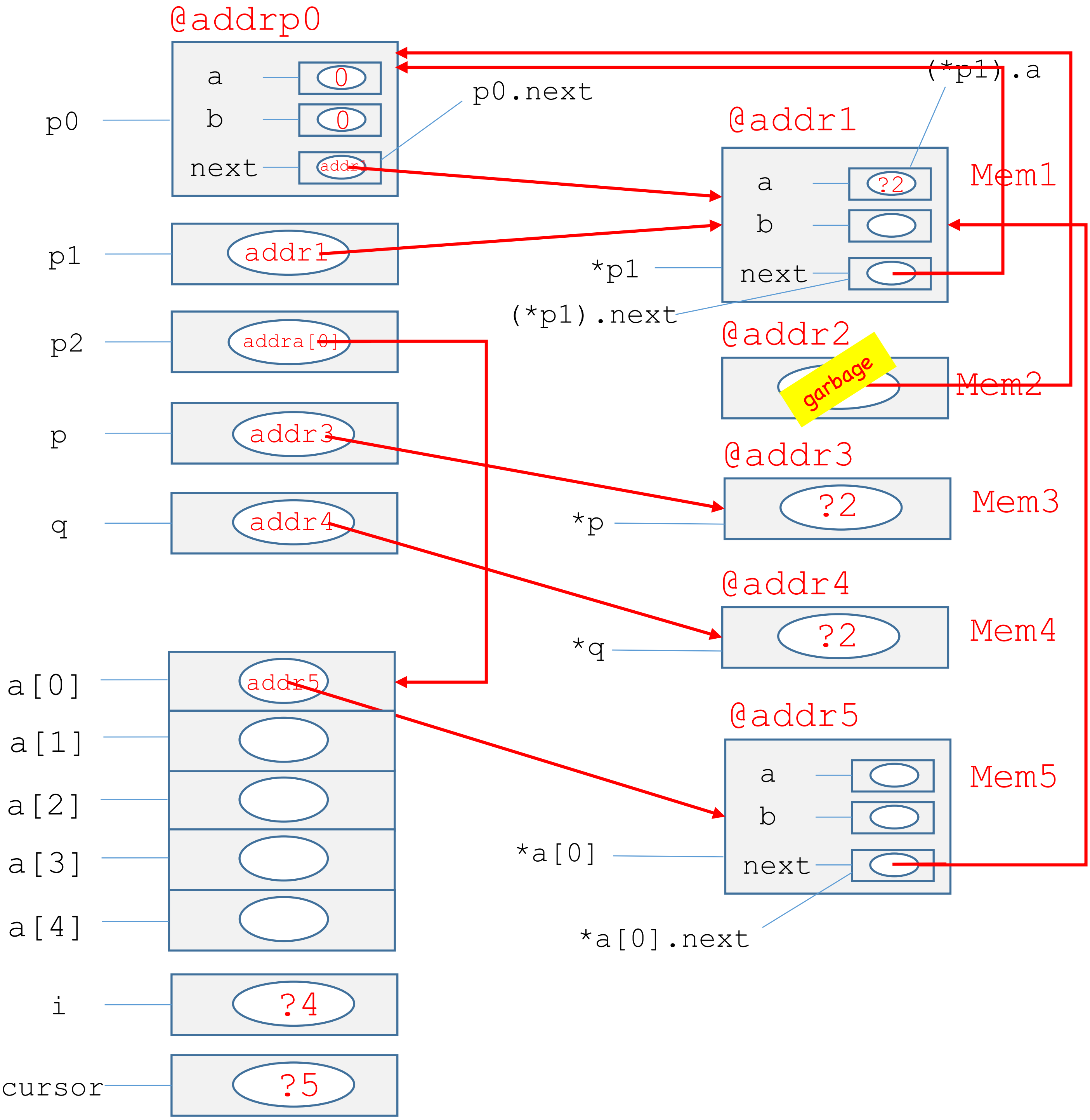


i is not initialized because it is a local variable

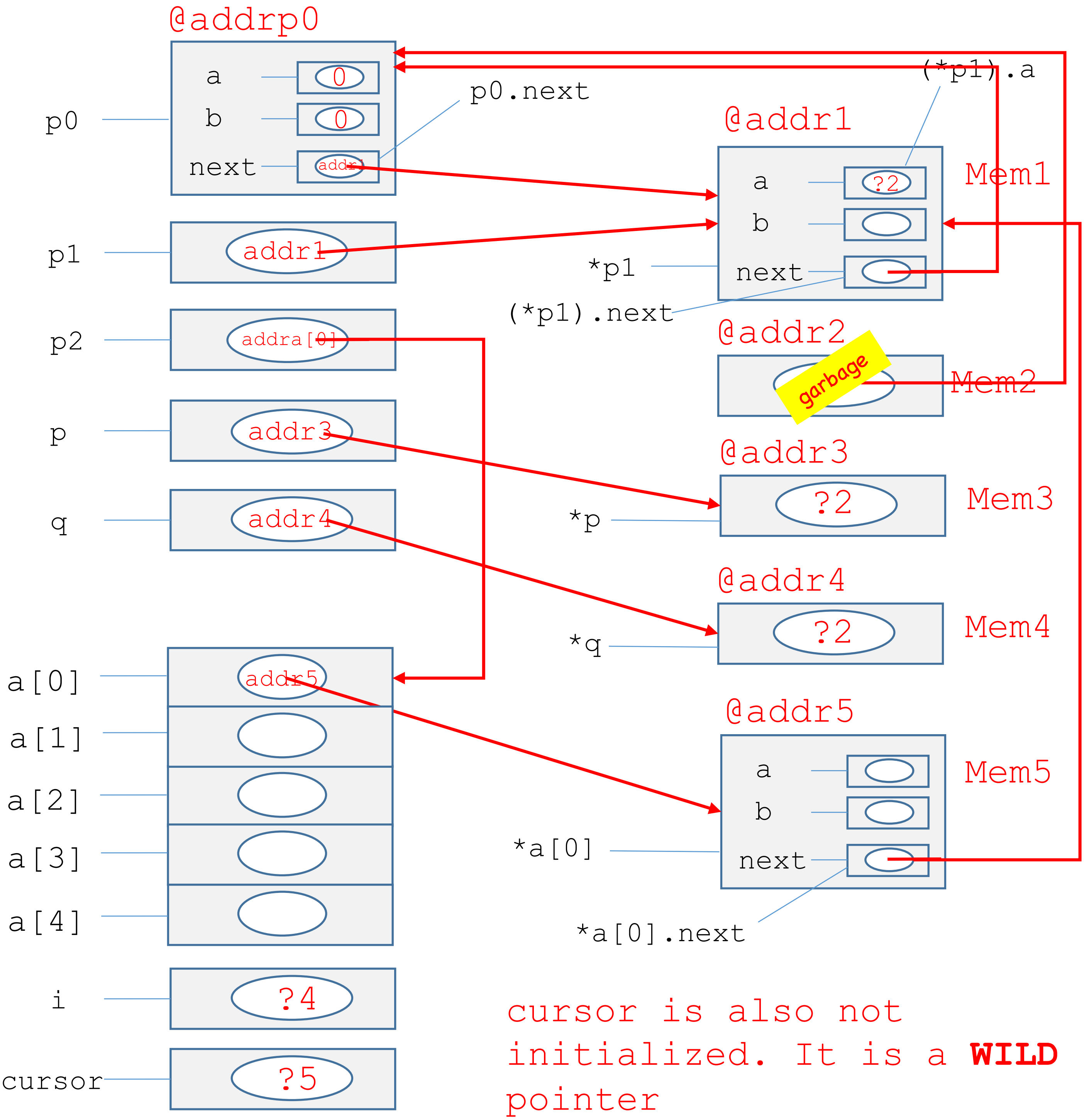
```
int main() {
    ...
    {
      struct T *a[5];
      ...
      int i;
      struct T* cursor;
```



```
int main() {
    ...
    {
       struct T *a[5];
       ...
       int i;
       struct T* cursor;
```

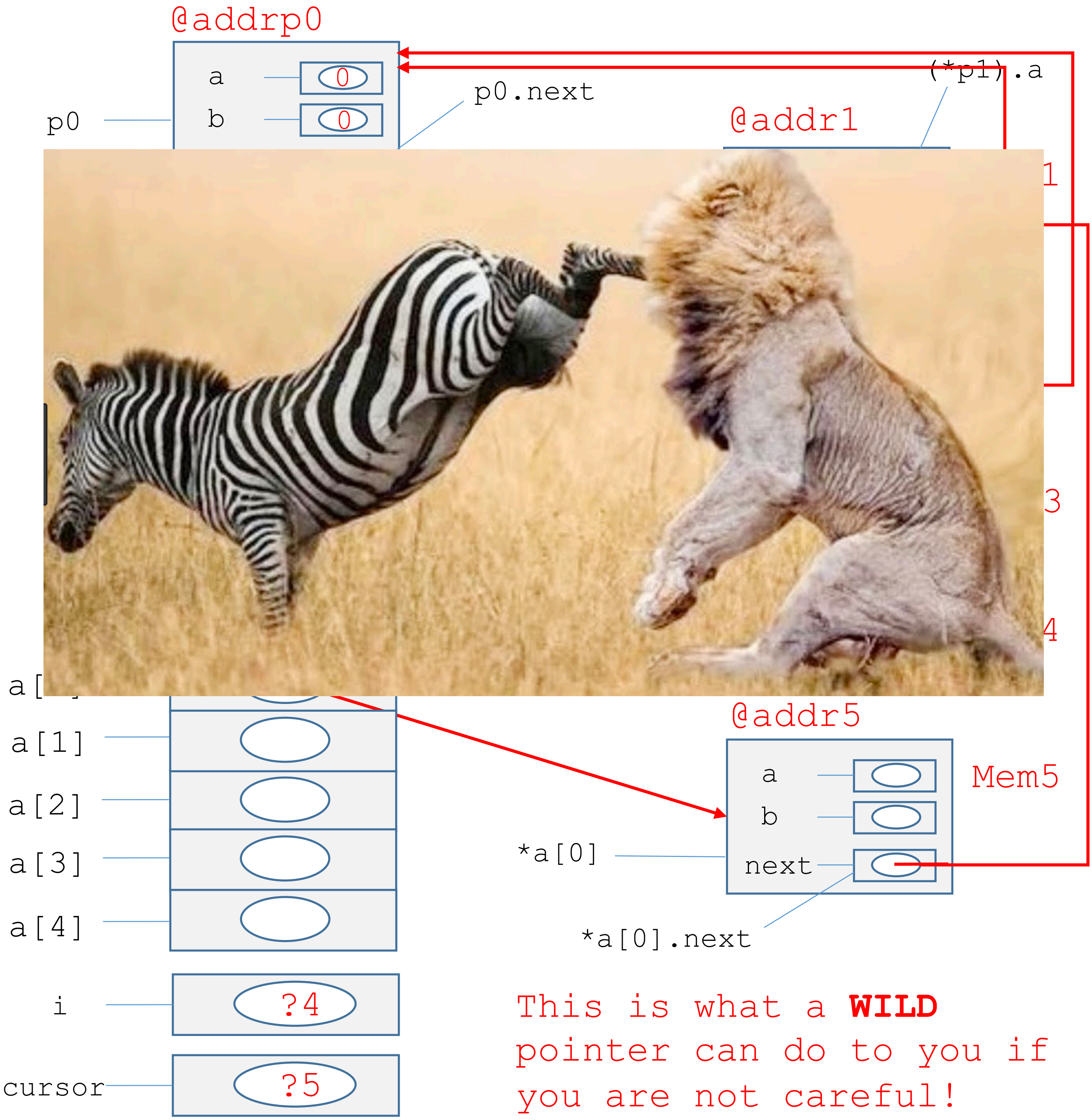


```
int main() {
    ...
    {
      struct T *a[5];
      ...
      int i;
      struct T* cursor;
```



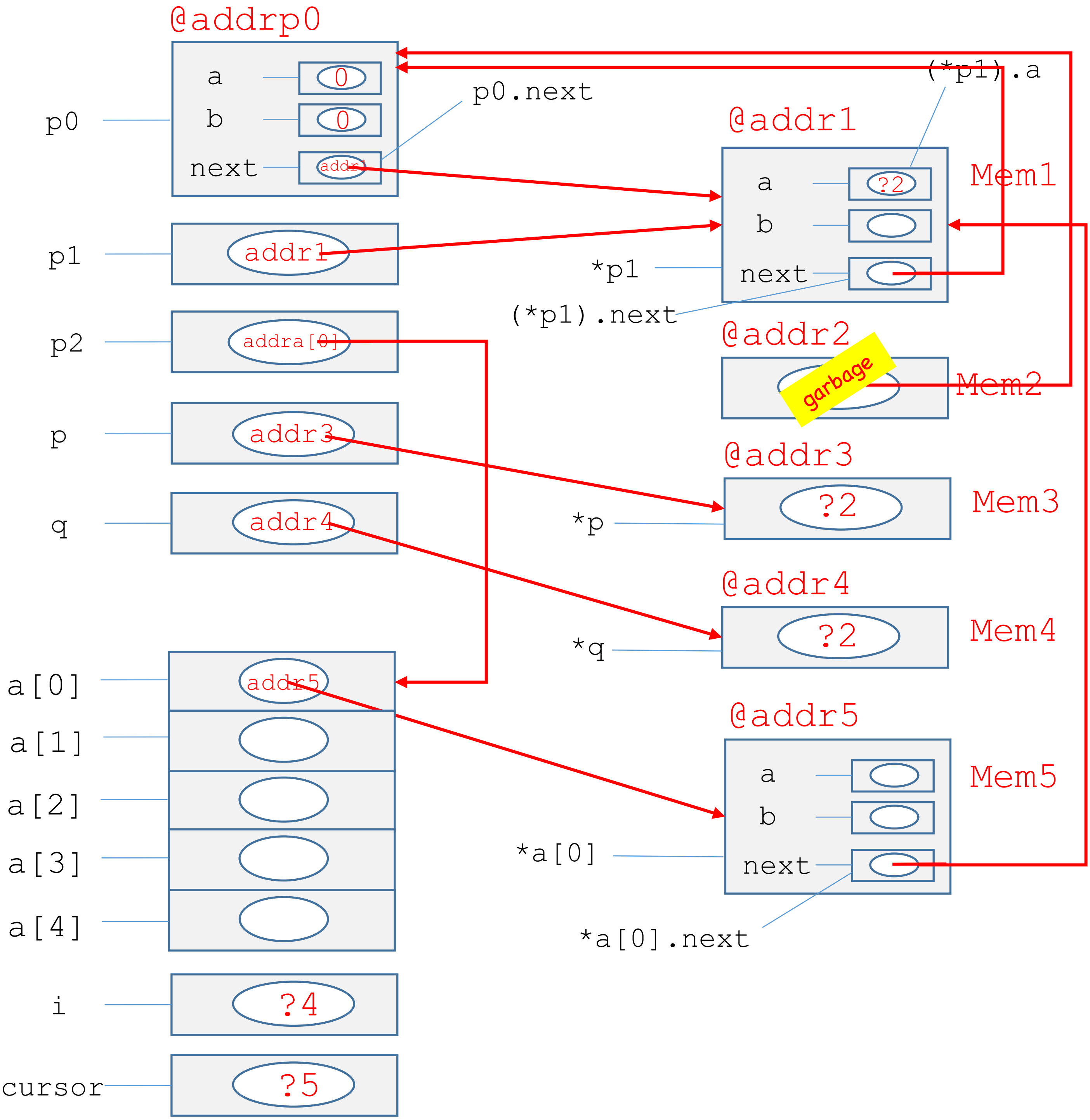
cursor is also not initialized. It is a **WILD** pointer

```
int main() {
    ...
    {
      struct T *a[5];
      ...
      int i;
      struct T* cursor;
```

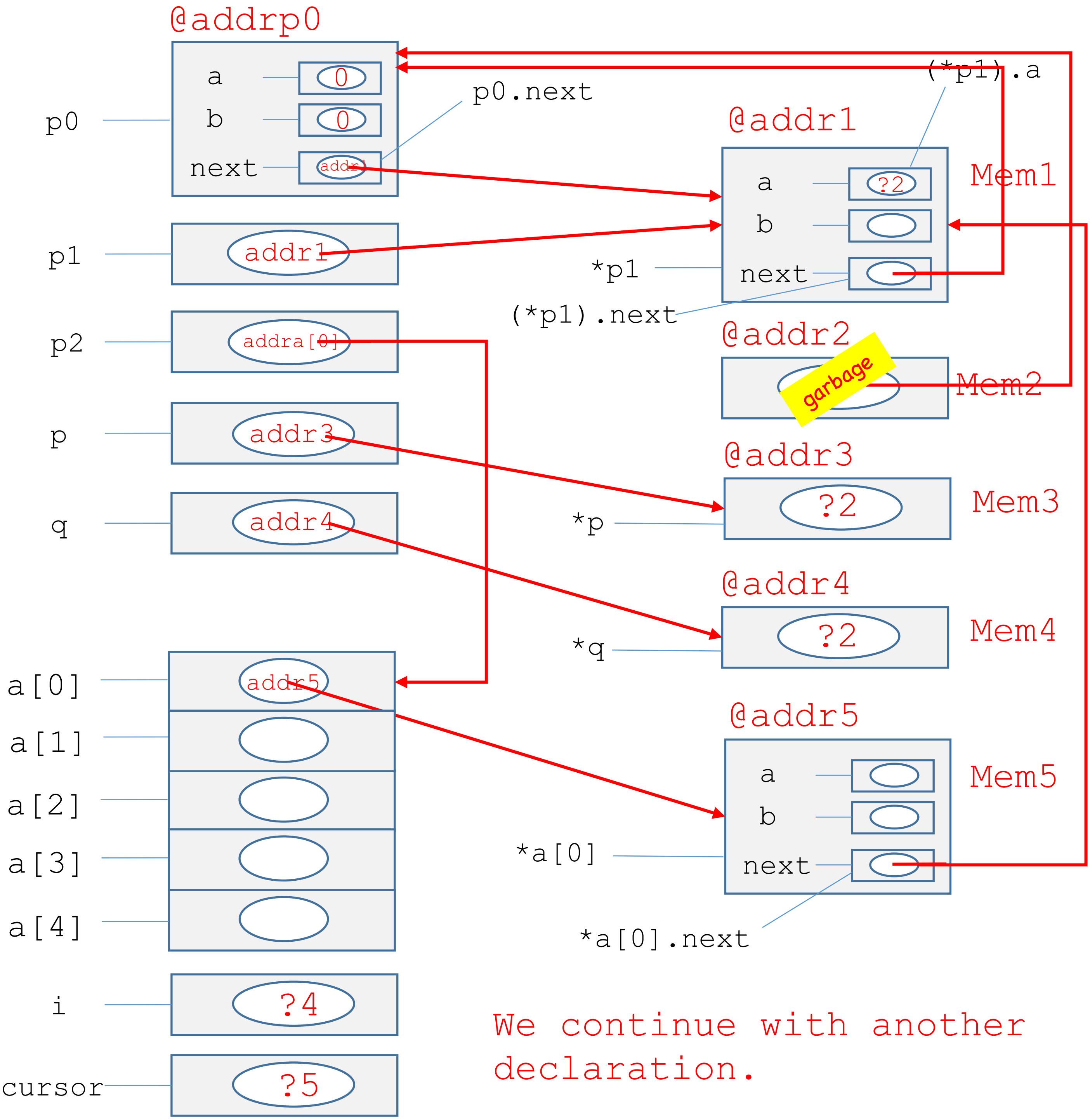


This is what a **WILD** pointer can do to you if you are not careful!

```
...
{
  struct T *a[5];
  ...
  int i;
  struct T* cursor;
  struct T **b[5];
```

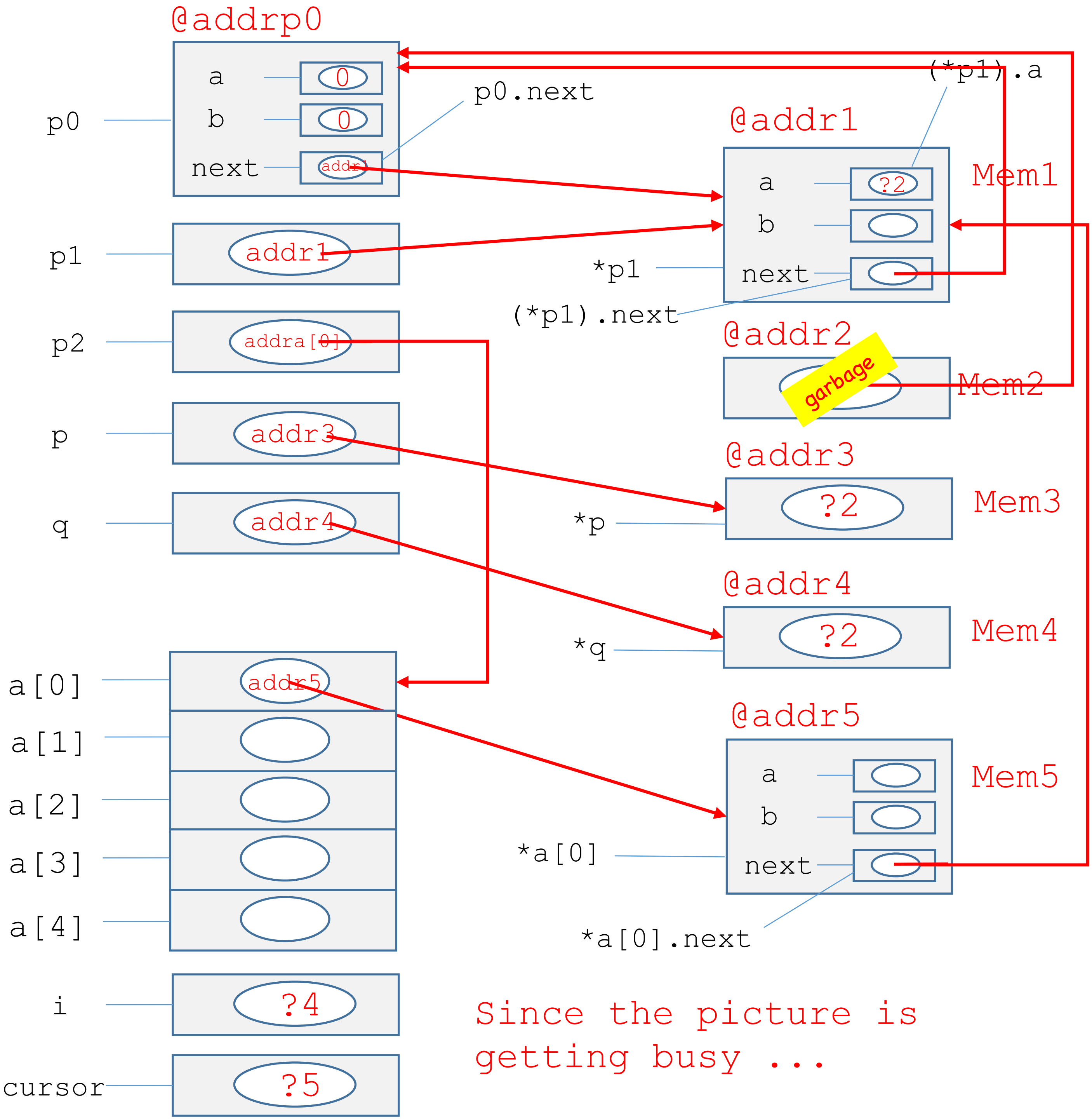


```
...
{
  struct T *a[5];
  ...
  int i;
  struct T* cursor;
  struct T **b[5];
```



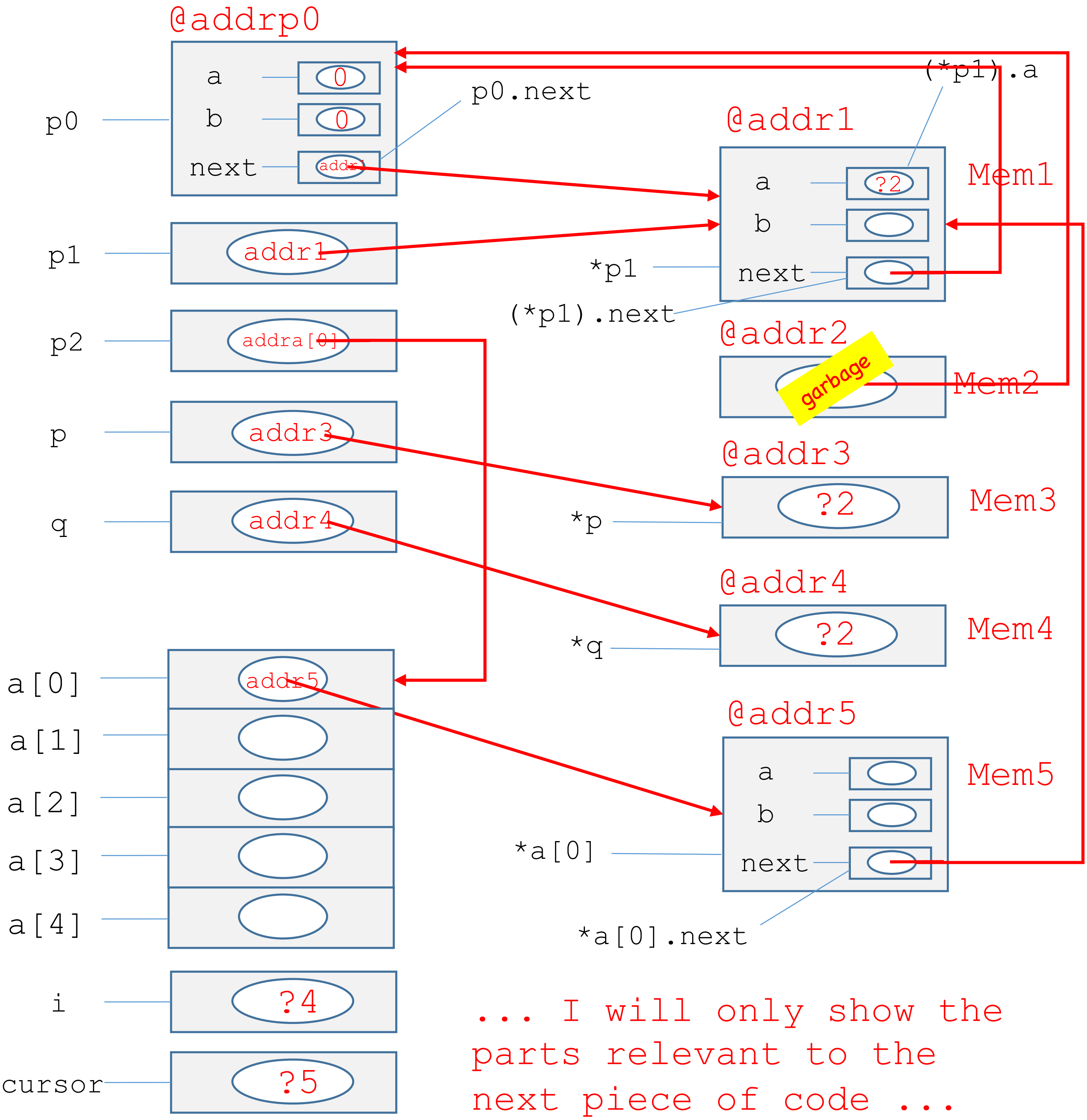
We continue with another declaration.

```
...
{
  struct T *a[5];
  ...
  int i;
  struct T* cursor;
  struct T **b[5];
```



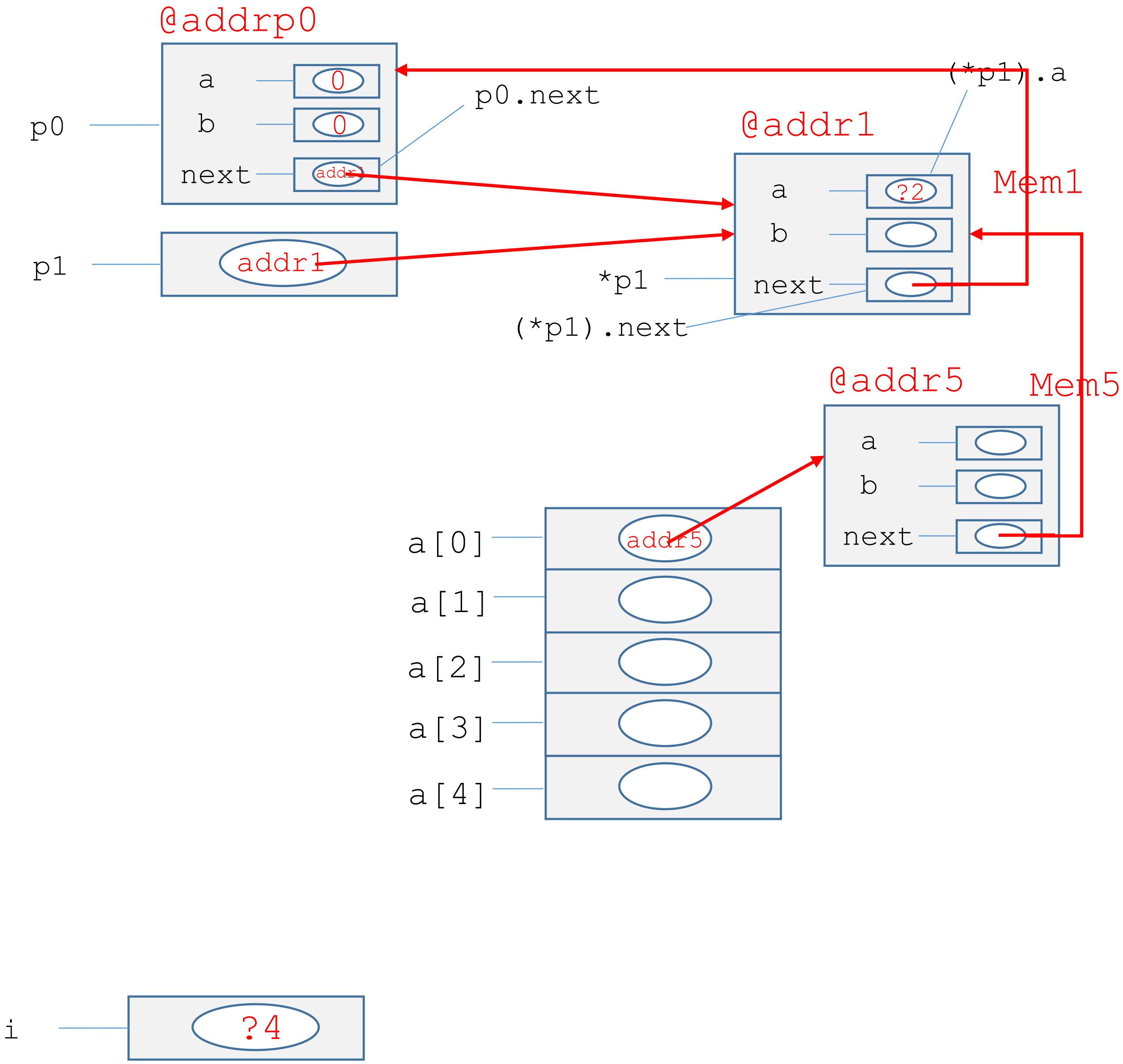
Since the picture is getting busy ...

```
...
{
  struct T *a[5];
  ...
  int i;
  struct T* cursor;
  struct T **b[5];
```

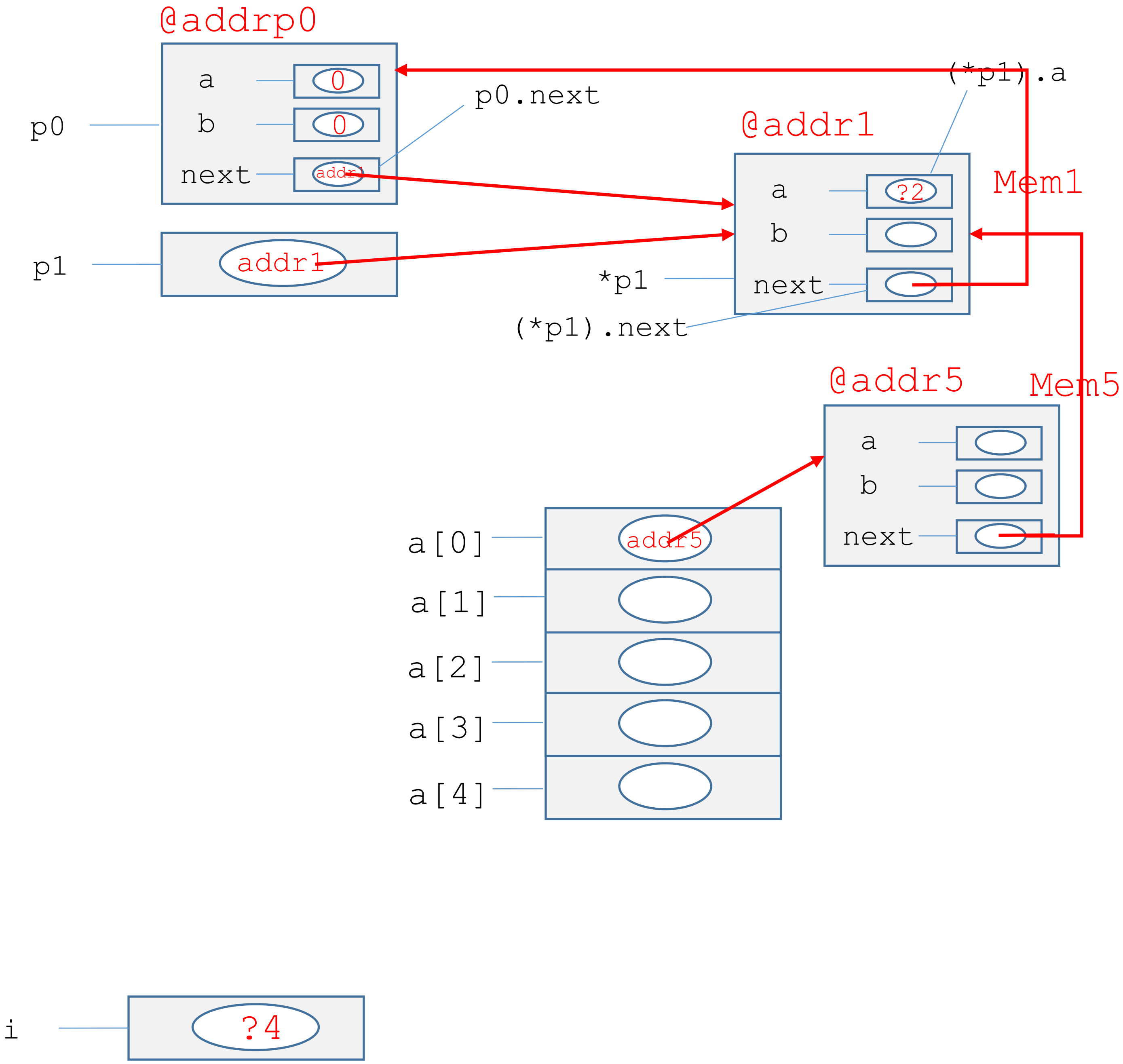


... I will only show the parts relevant to the next piece of code ...

```
...
{
  struct T *a[5];
  ...
  int i;
  struct T* cursor;
  struct T **b[5];
```

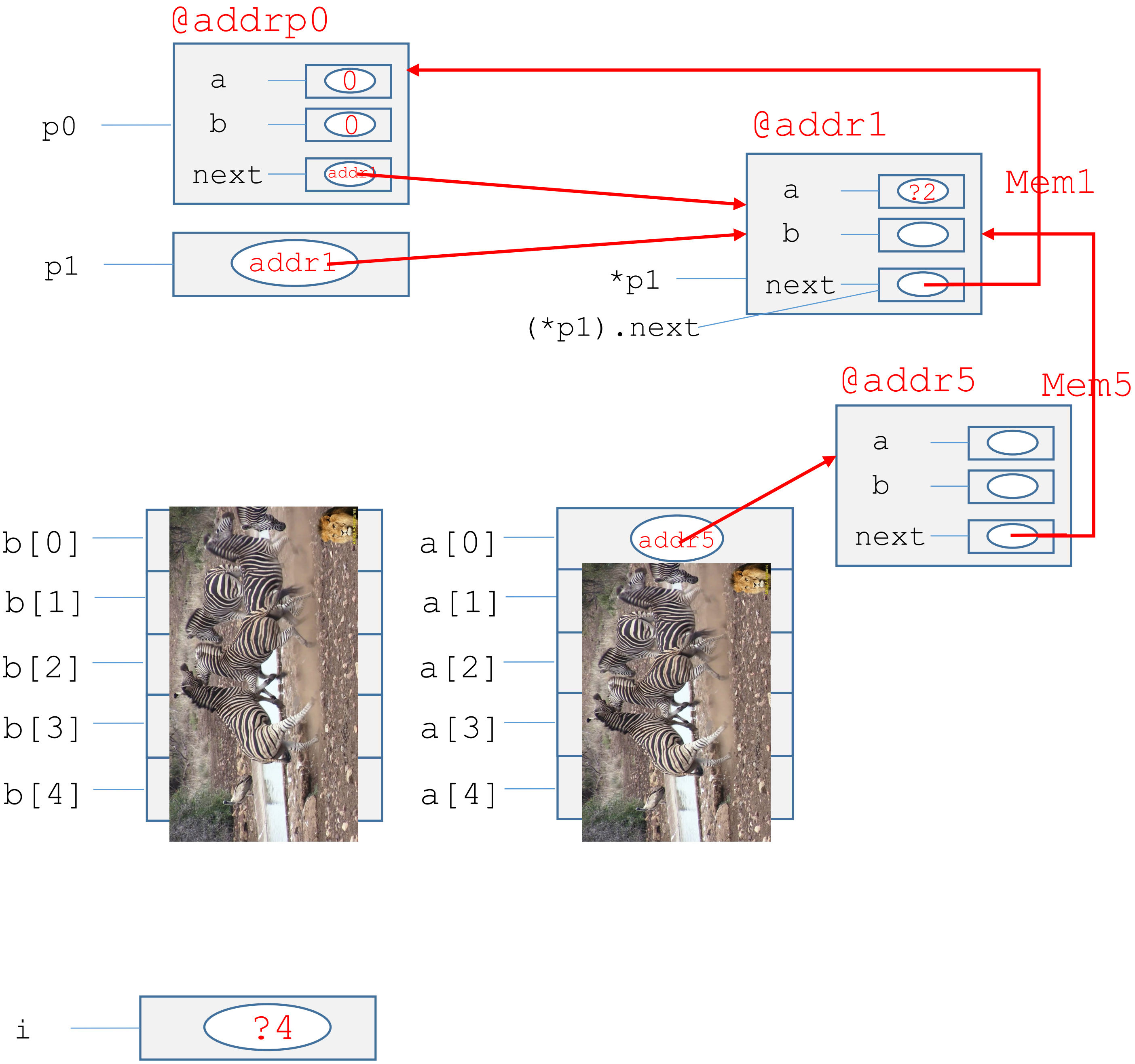


```
...
{
   struct T *a[5];
   ...
   int i;
   struct T* cursor;
   struct T **b[5];
```

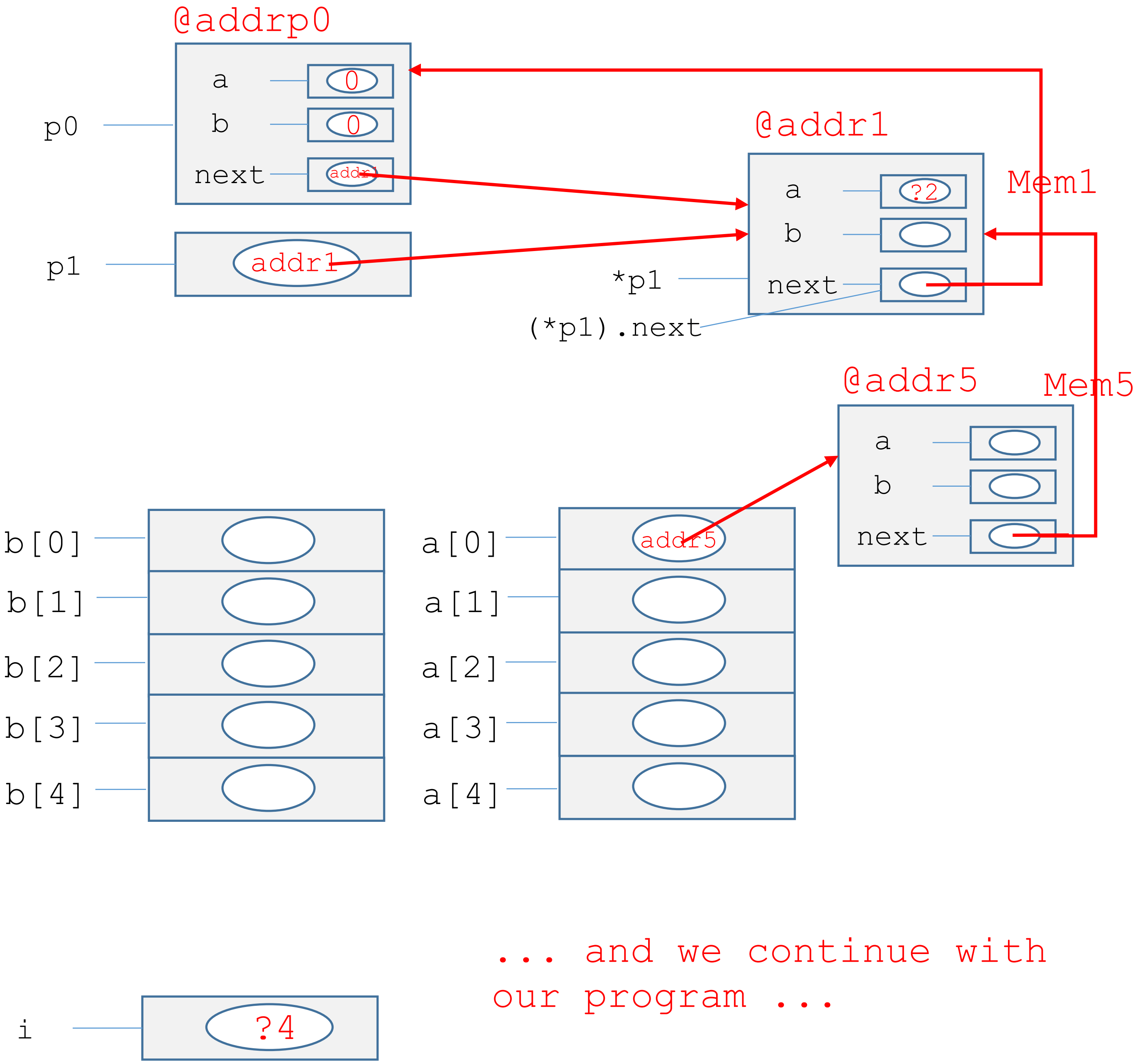


@addrp0
p0
a
0
b
0
next
addr1
p0.next
p1
addr1
@addr1
(*p1).a
a
?2
b
next
*p1
(*p1).next
Mem1
@addr5
Mem5
a
b
next
a[0]
addr5
a[1]
a[2]
a[3]
a[4]
i
?4

```
...
{
  struct T *a[5];
  ...
  int i;
  struct T* cursor;
  struct T **b[5];
```

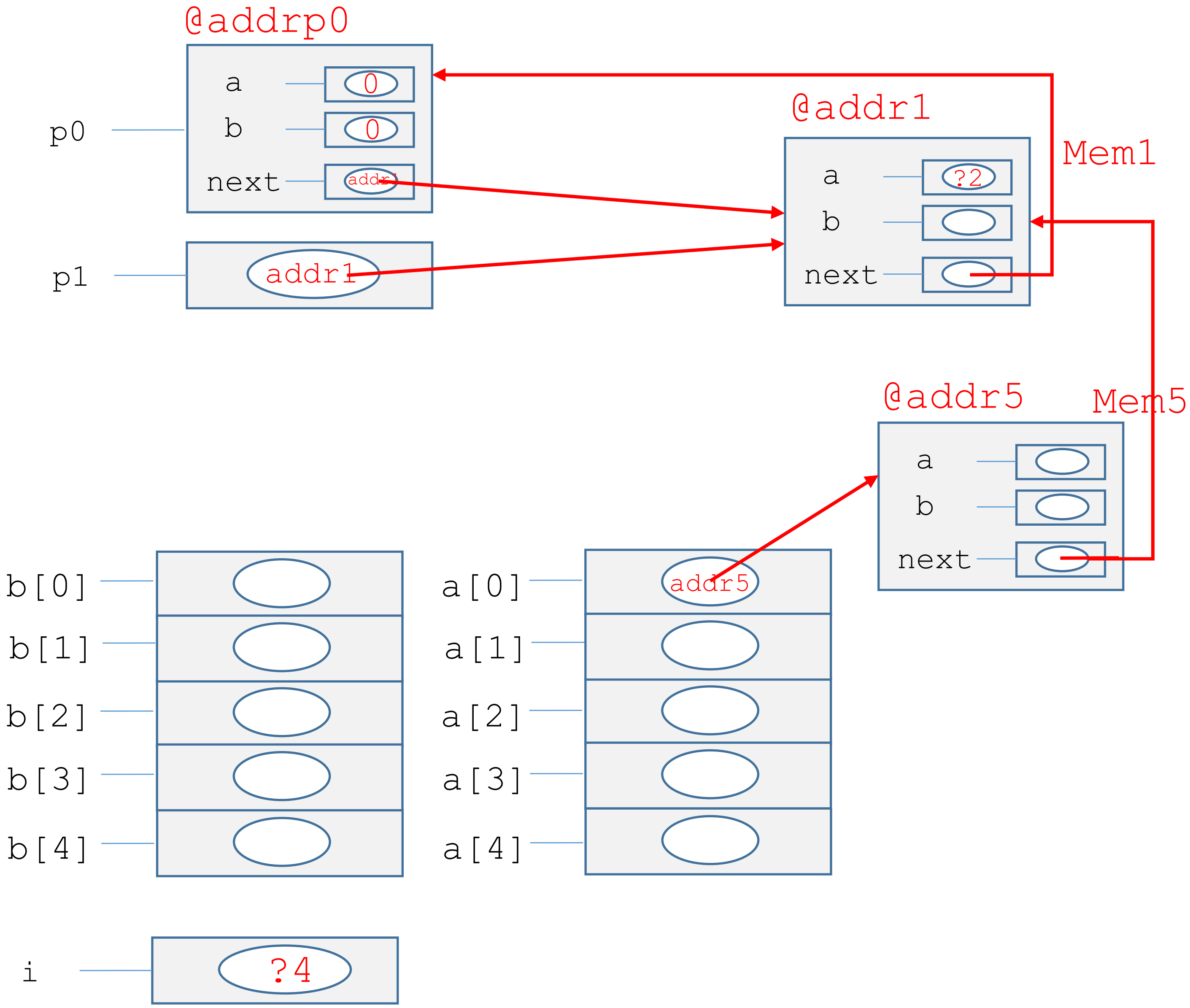


```
...
{
  struct T *a[5];
  ...
  int i;
  struct T* cursor;
  struct T **b[5];
```

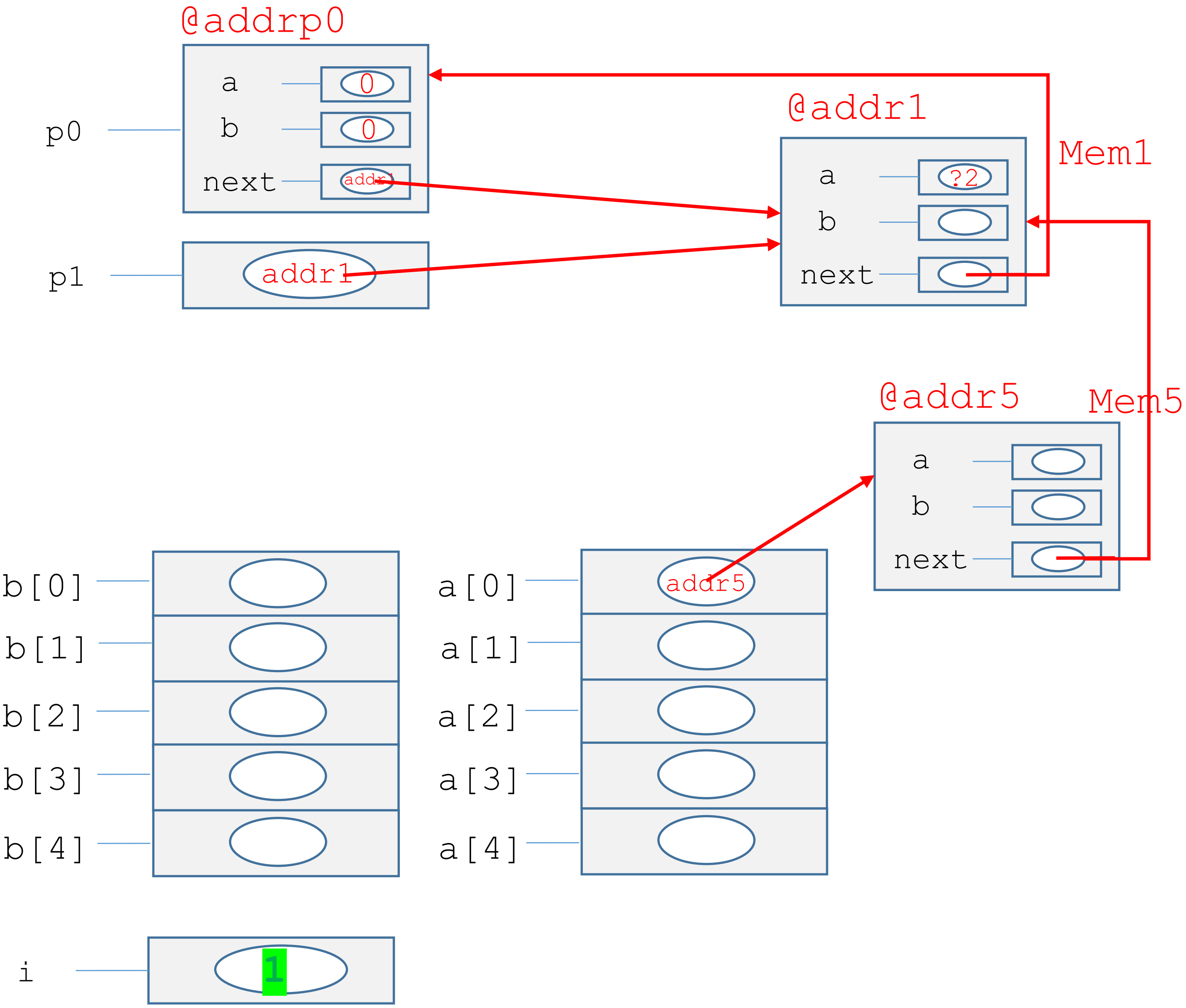


... and we continue with our program ...

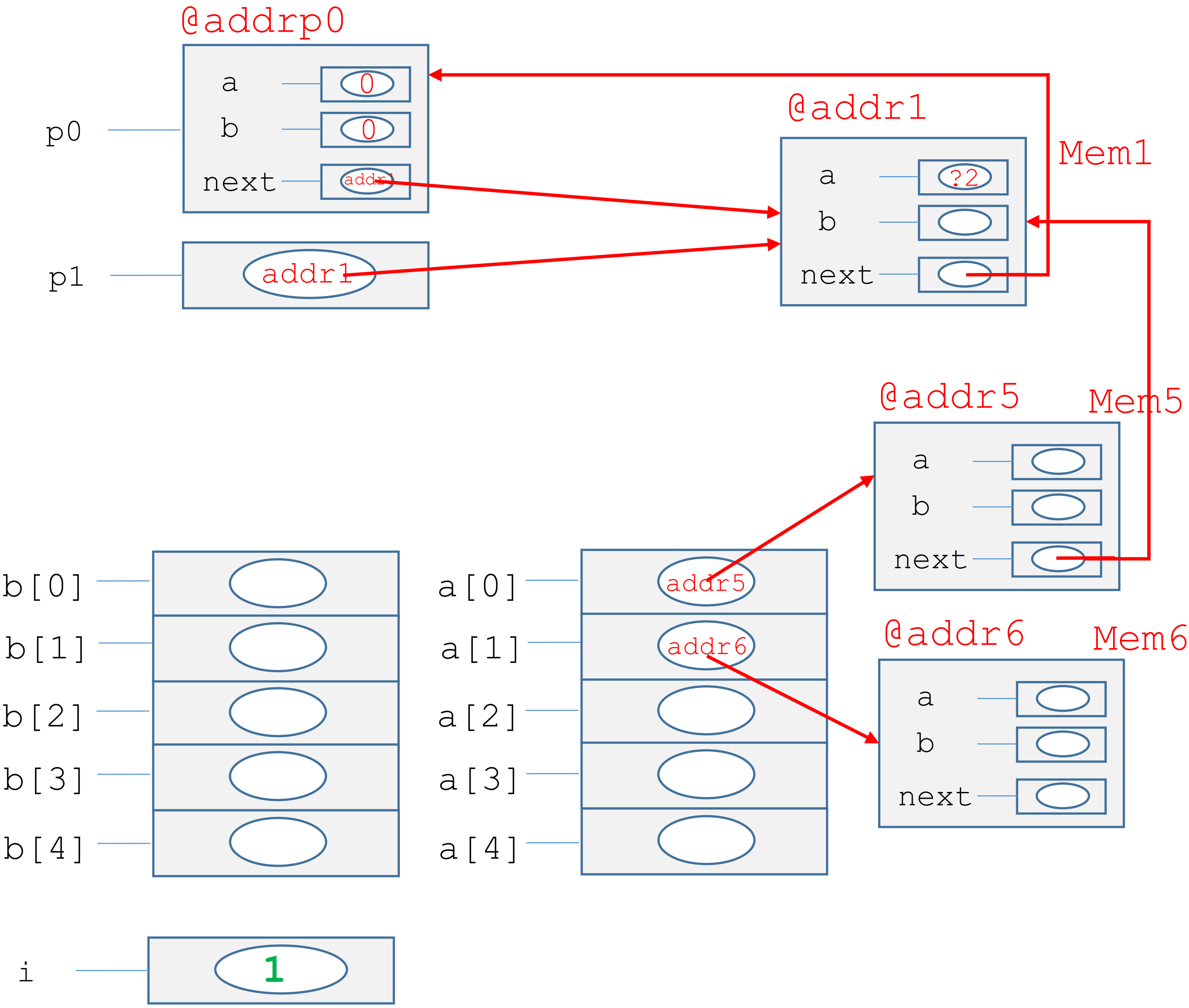
```
for (i = 1; i < 4; i++)
{
   a[i] = (struct T*) malloc(sizeof(struct T));
   (*a[i]).next = a[i-1];
   (*a[i]).a = i;
   b[i] = &a[i];
   (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```



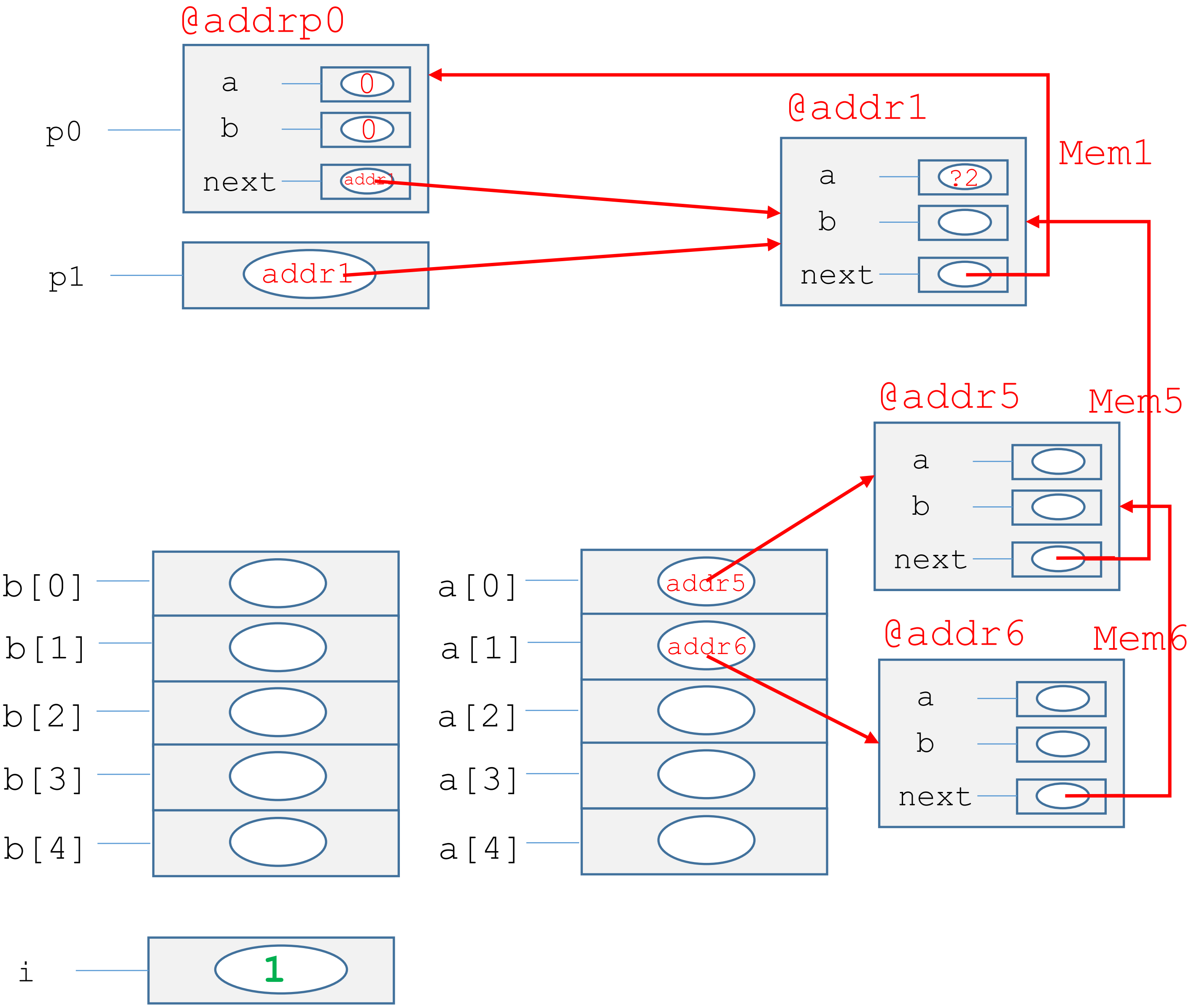
```
for (i = 1; i < 4; i++)
{
   a[i] = (struct T*) malloc(sizeof(struct T));
   (*a[i]).next = a[i-1];
   (*a[i]).a = i;
   b[i] = &a[i];
   (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```



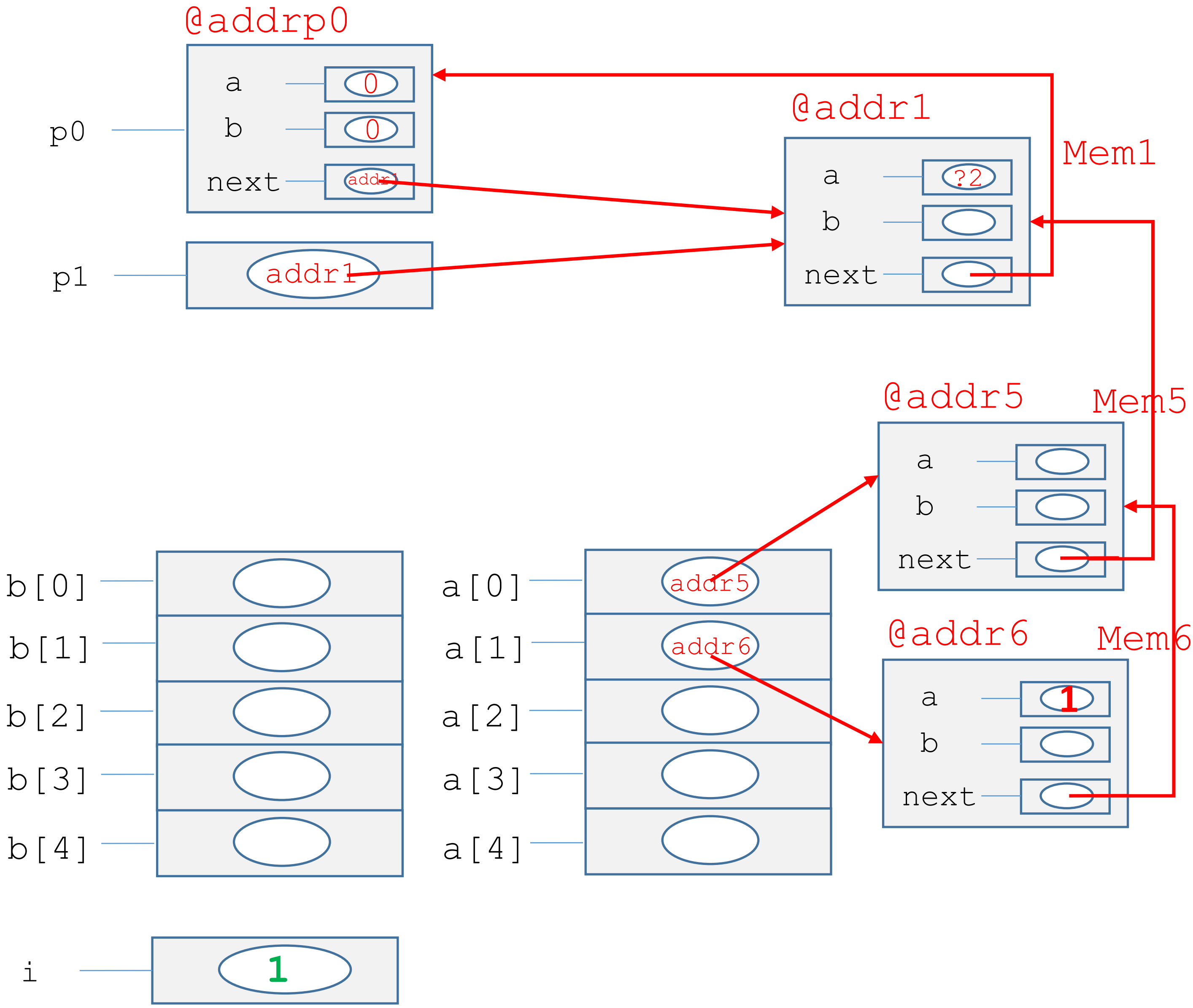
```
for (i = 1; i < 4; i++)
{
   a[i] = (struct T*) malloc(sizeof(struct T));
   (*a[i]).next = a[i-1];
   (*a[i]).a = i;
   b[i] = &a[i];
   (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```



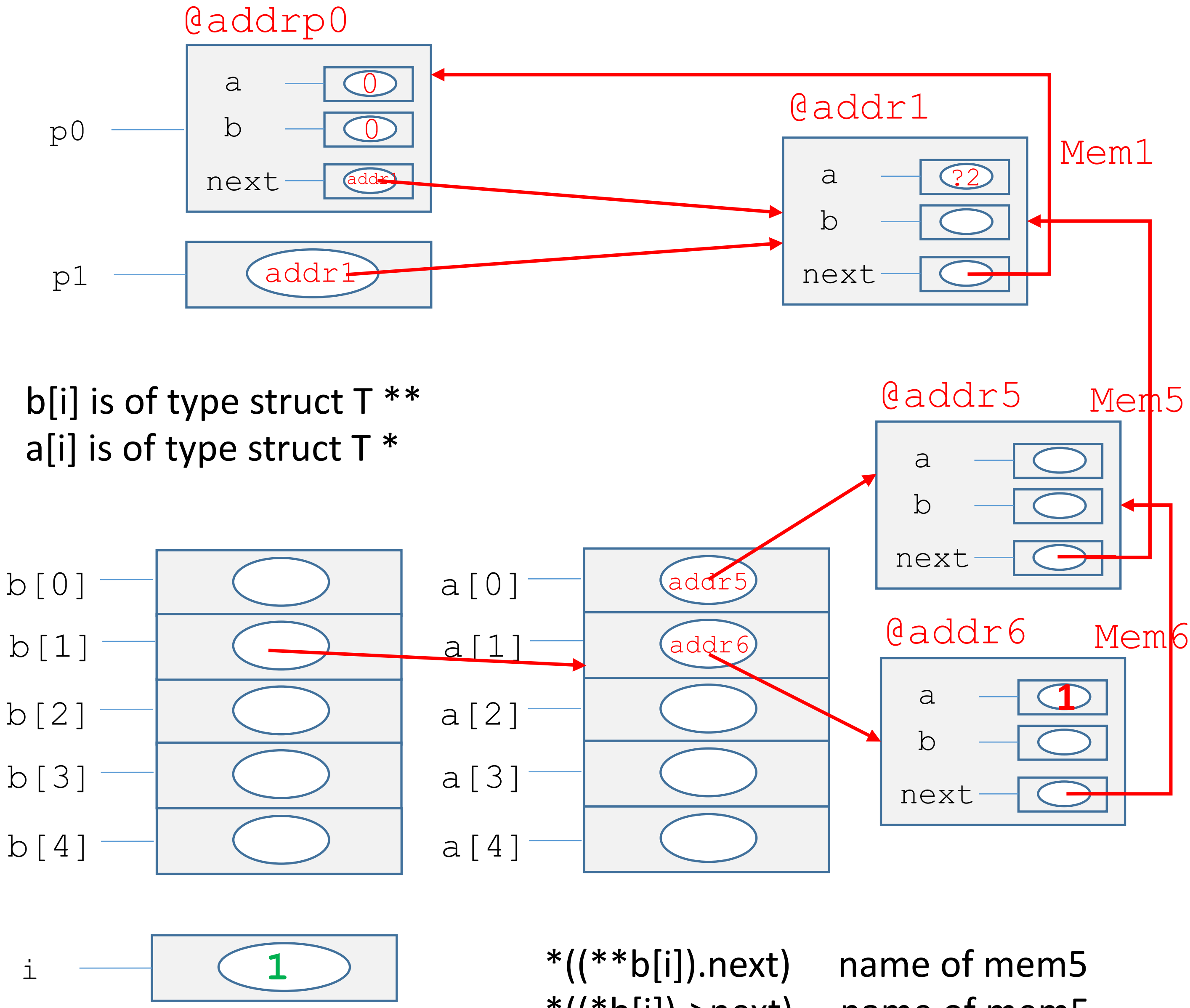
```
for (i = 1; i < 4; i++)
{
   a[i] = (struct T*) malloc(sizeof(struct T));
   (*a[i]).next = a[i-1];      // a[i]->next
   (*a[i]).a = i;              //a[i]->a
   b[i] = &a[i];
   (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```



```
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];
```



```
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];
```



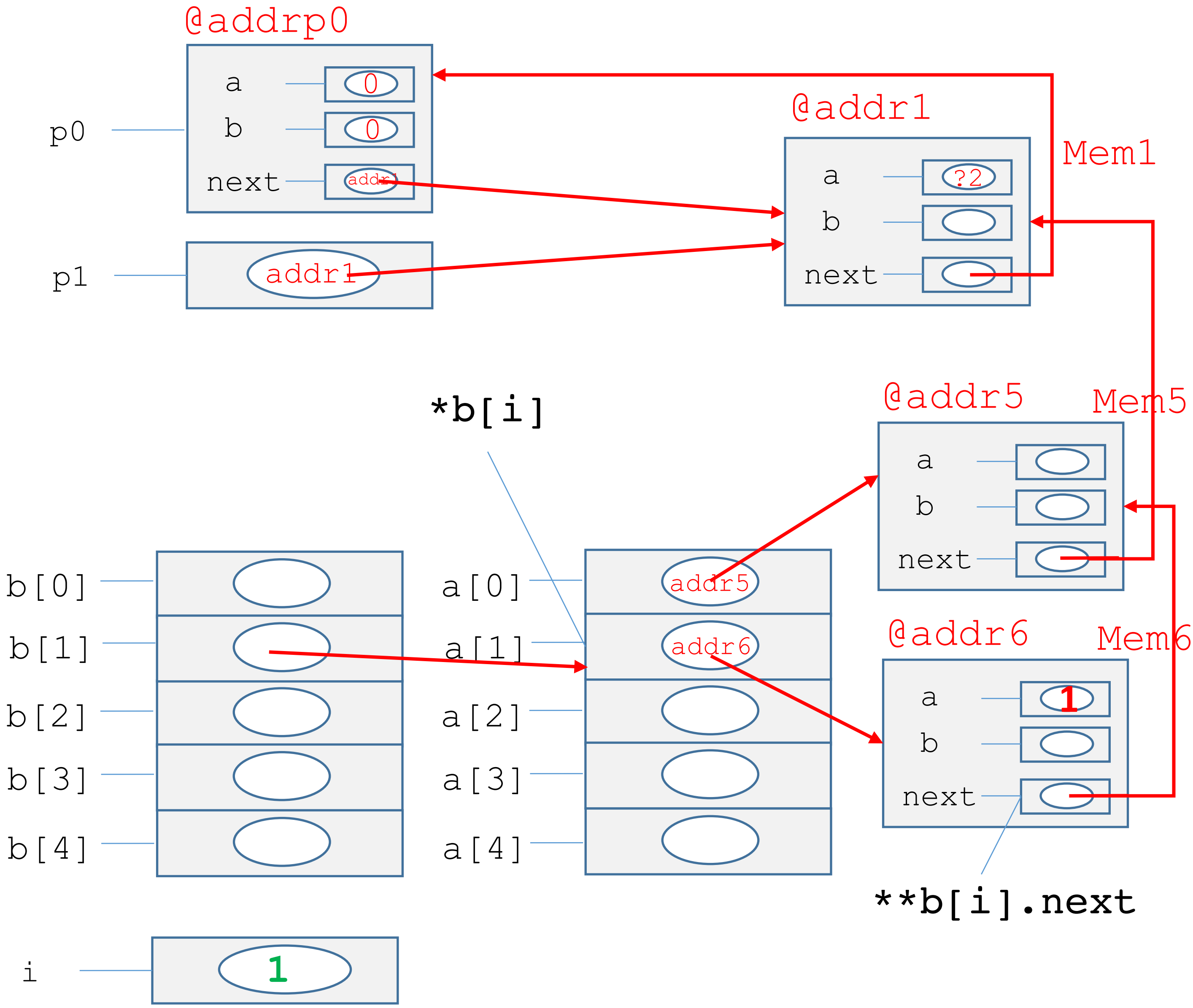
b[i] is of type struct T **

a[i] is of type struct T *

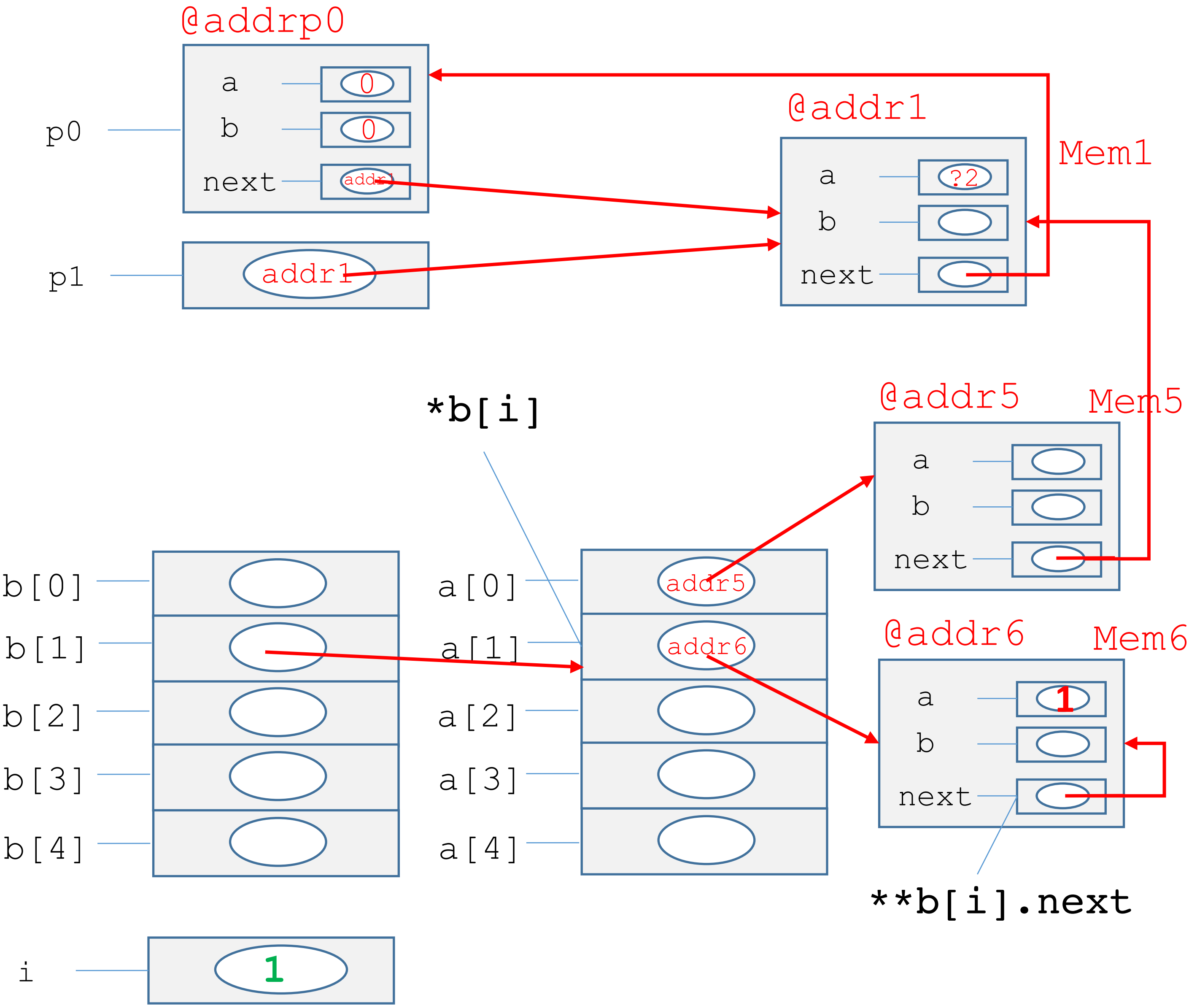
*((**b[i]).next) name of mem5

*((*b[i])->next) name of mem5

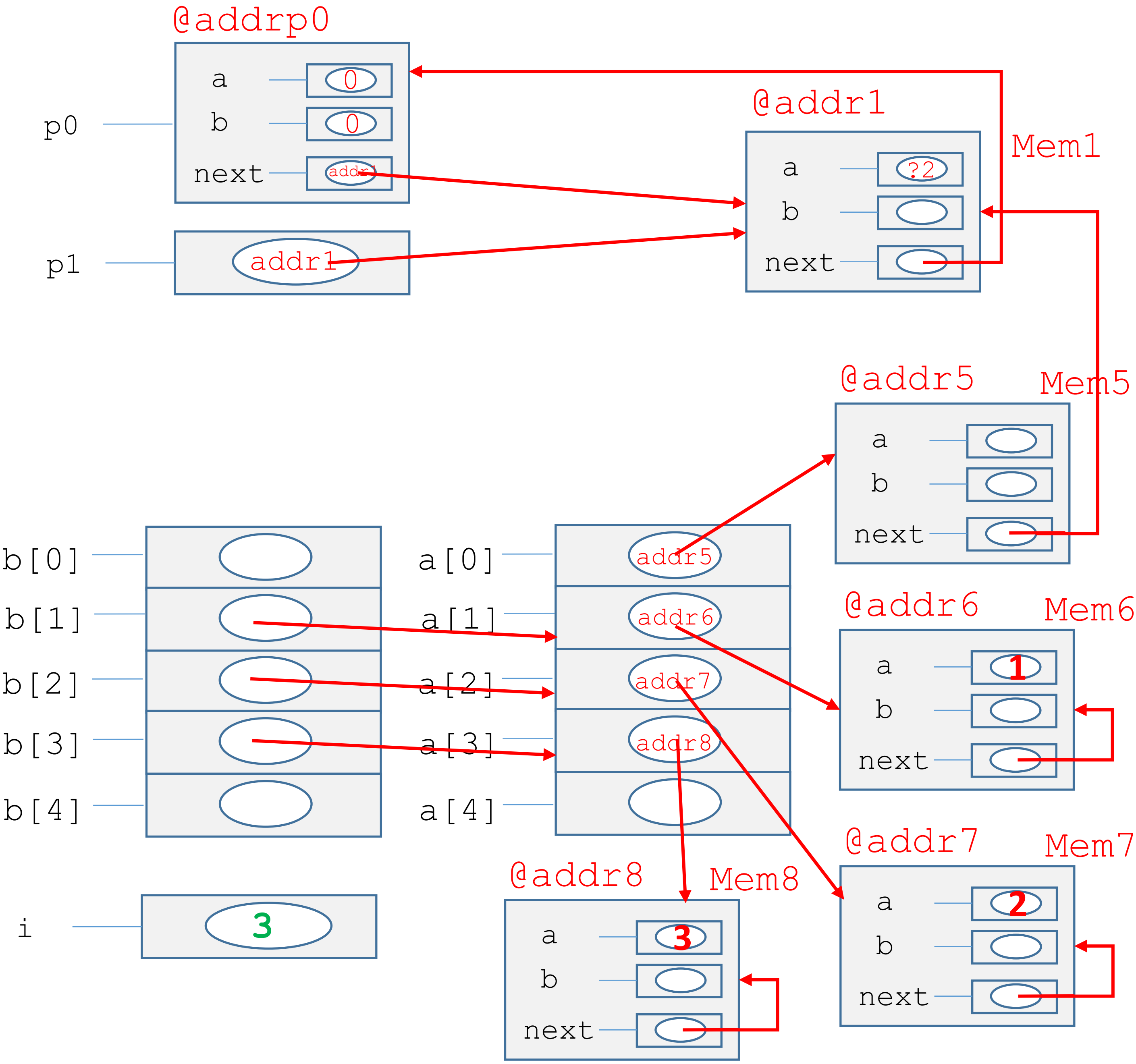
```
for (i = 1; i < 4; i++)
{
   a[i] = (struct T*) malloc(sizeof(struct T));
   (*a[i]).next = a[i-1];
   (*a[i]).a = i;
   b[i] = &a[i];
   (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```



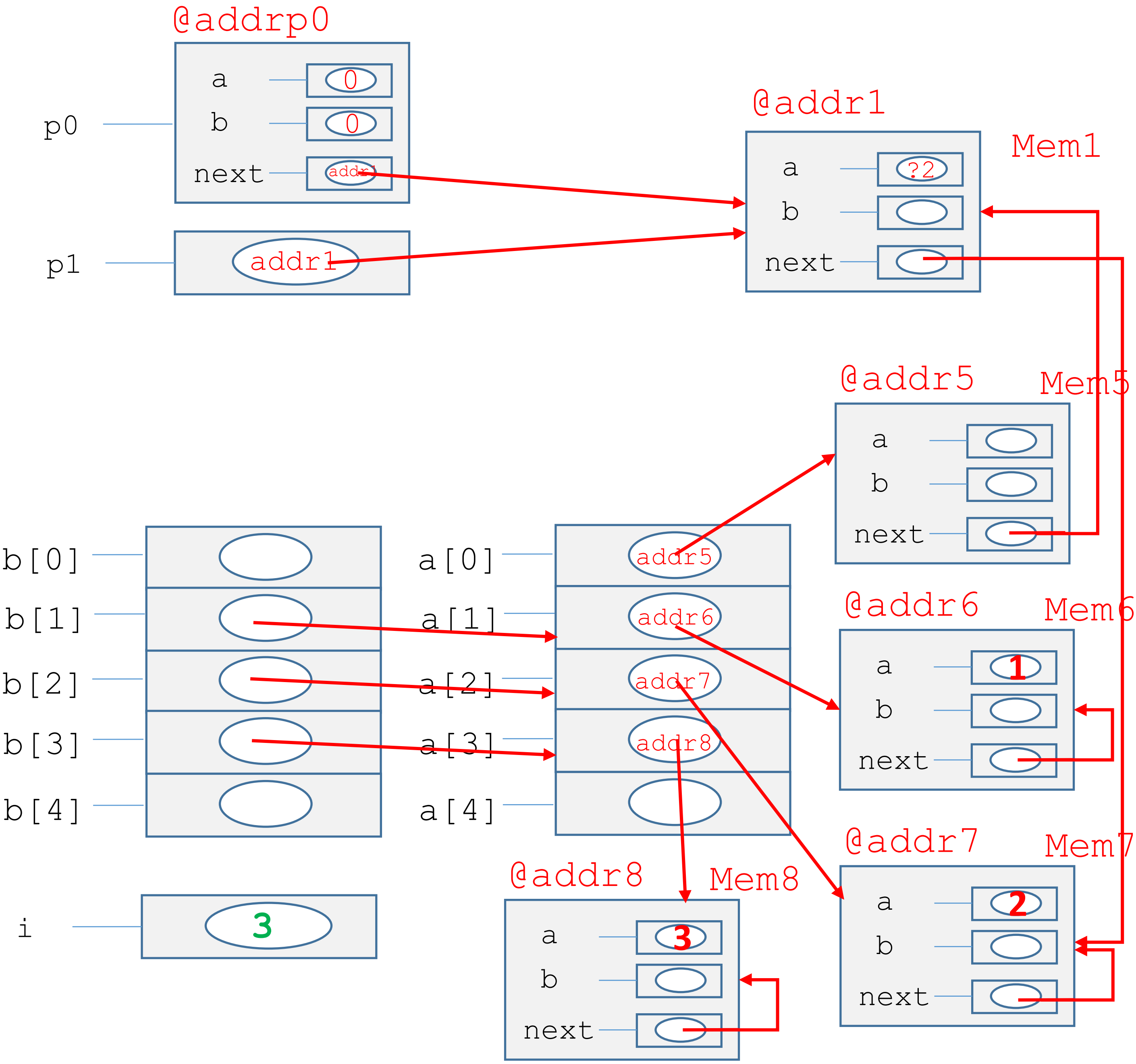
```
for (i = 1; i < 4; i++)
{
   a[i] = (struct T*) malloc(sizeof(struct T));
   (*a[i]).next = a[i-1];
   (*a[i]).a = i;
   b[i] = &a[i];
   (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```



```
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];
}       // at the end of the loop
(*p1).next = a[2];
```

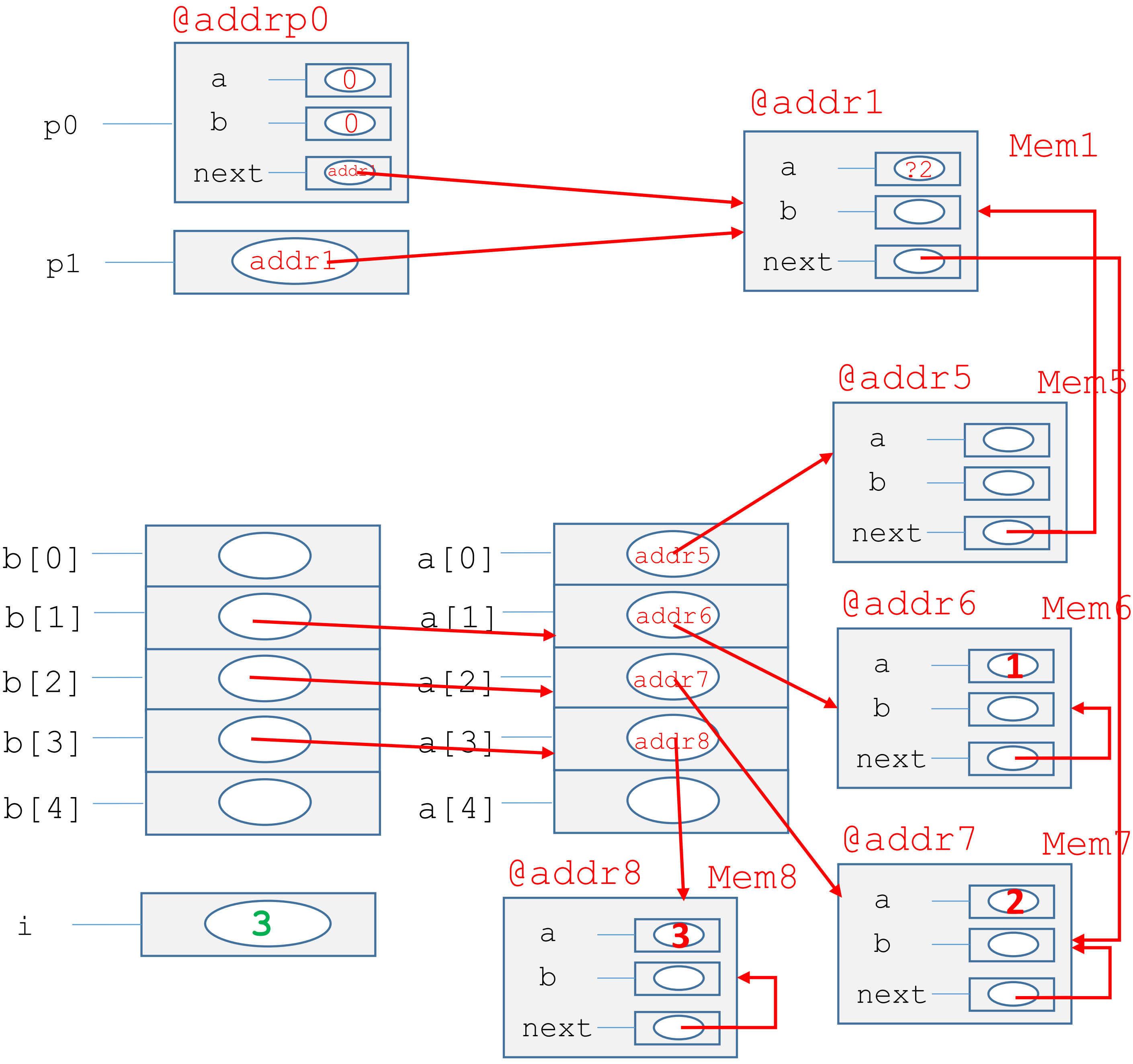


```
for (i = 1; i < 4; i++)
{
    a[i] = (struct T*) malloc(sizeof(struct T));
    (*a[i]).next = a[i-1];
    (*a[i]).a = i;
    b[i] = &a[i];
    (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```

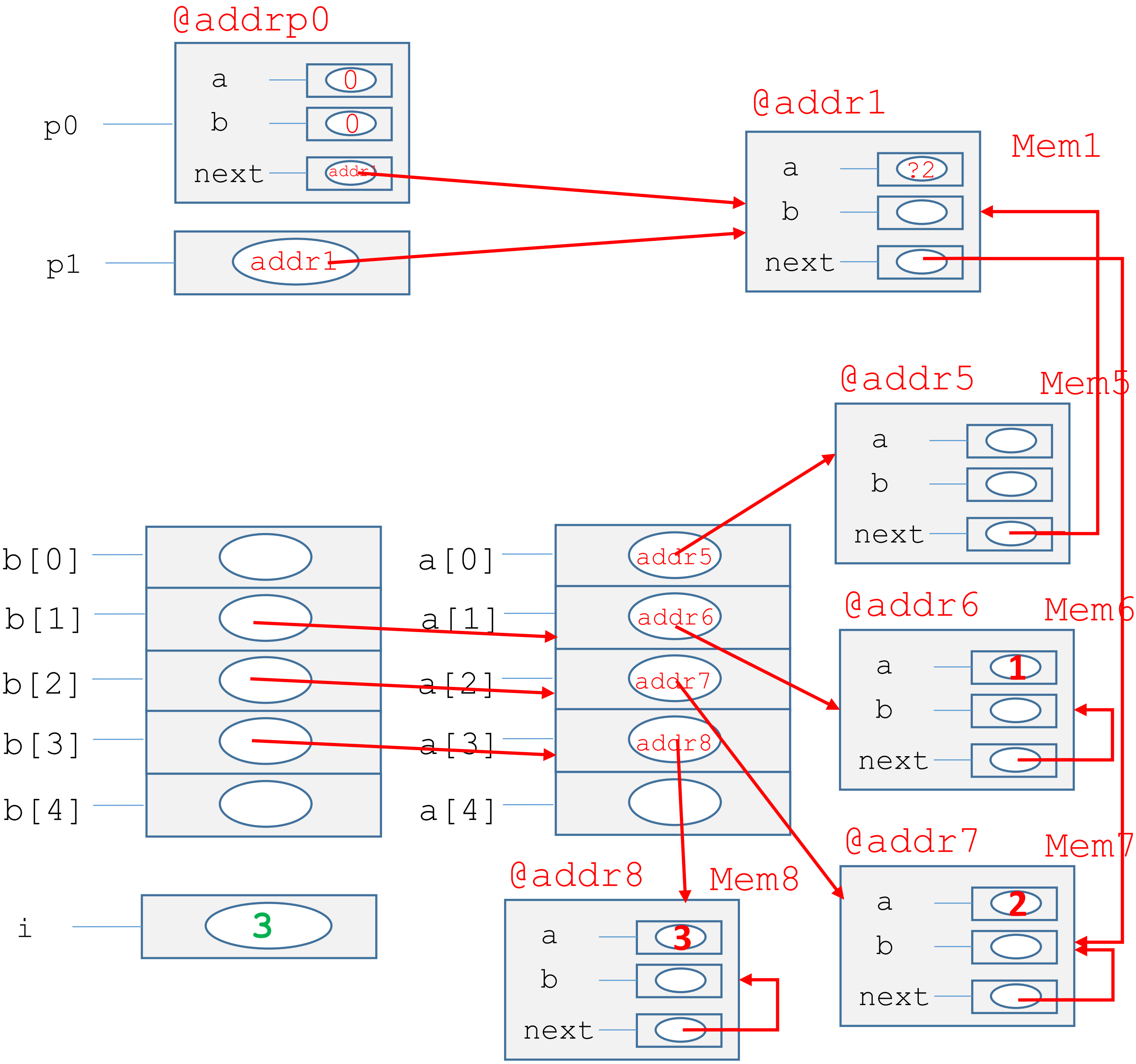


```
for (i = 1; i < 4; i++)
{
   a[i] = (struct T*) malloc(sizeof(struct T));
   (*a[i]).next = a[i-1];
   (*a[i]).a = i;
   b[i] = &a[i];
   (**b[i]).next = *b[i];
}
(*p1).next = a[2];
```

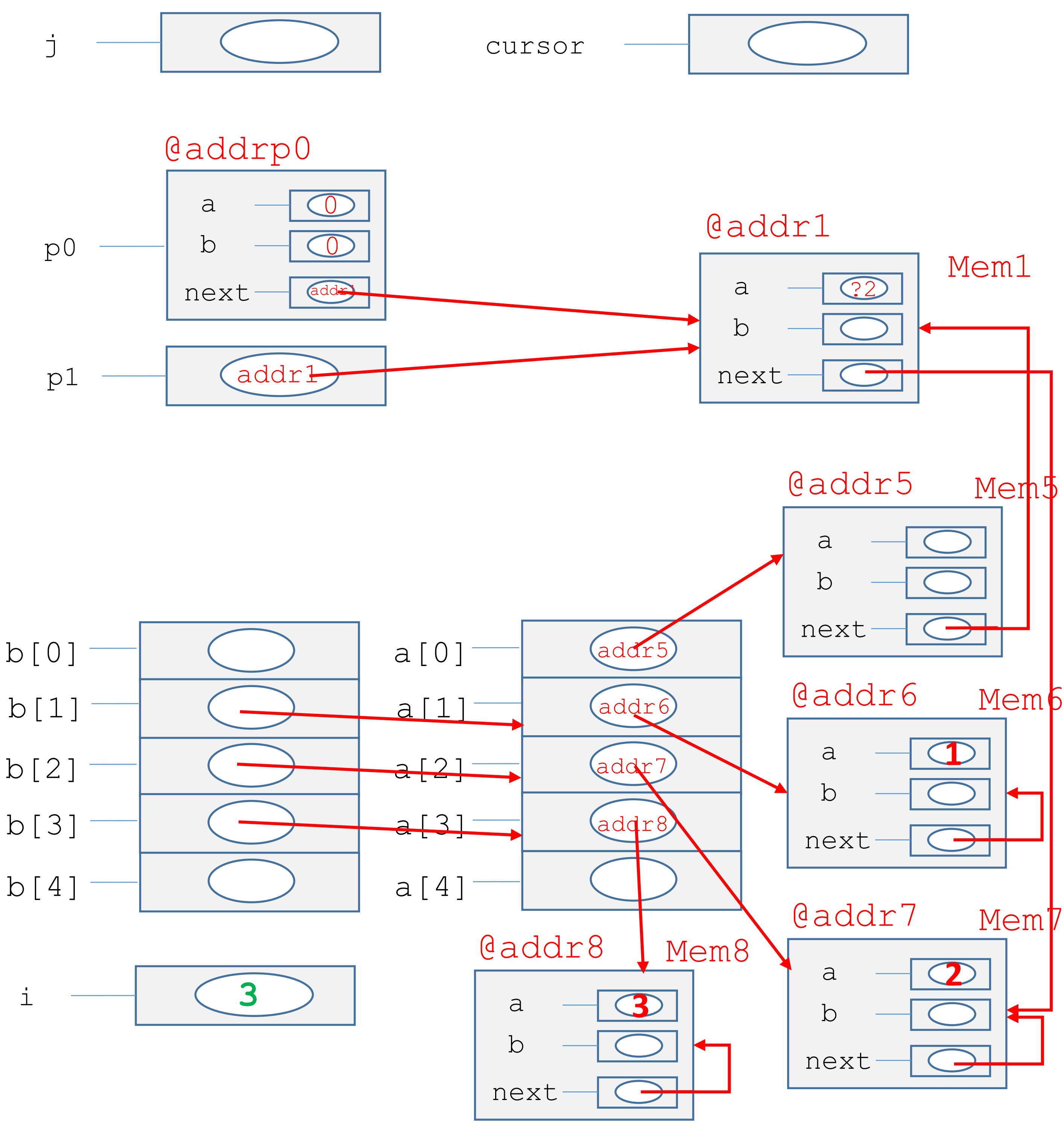
... we continue ...



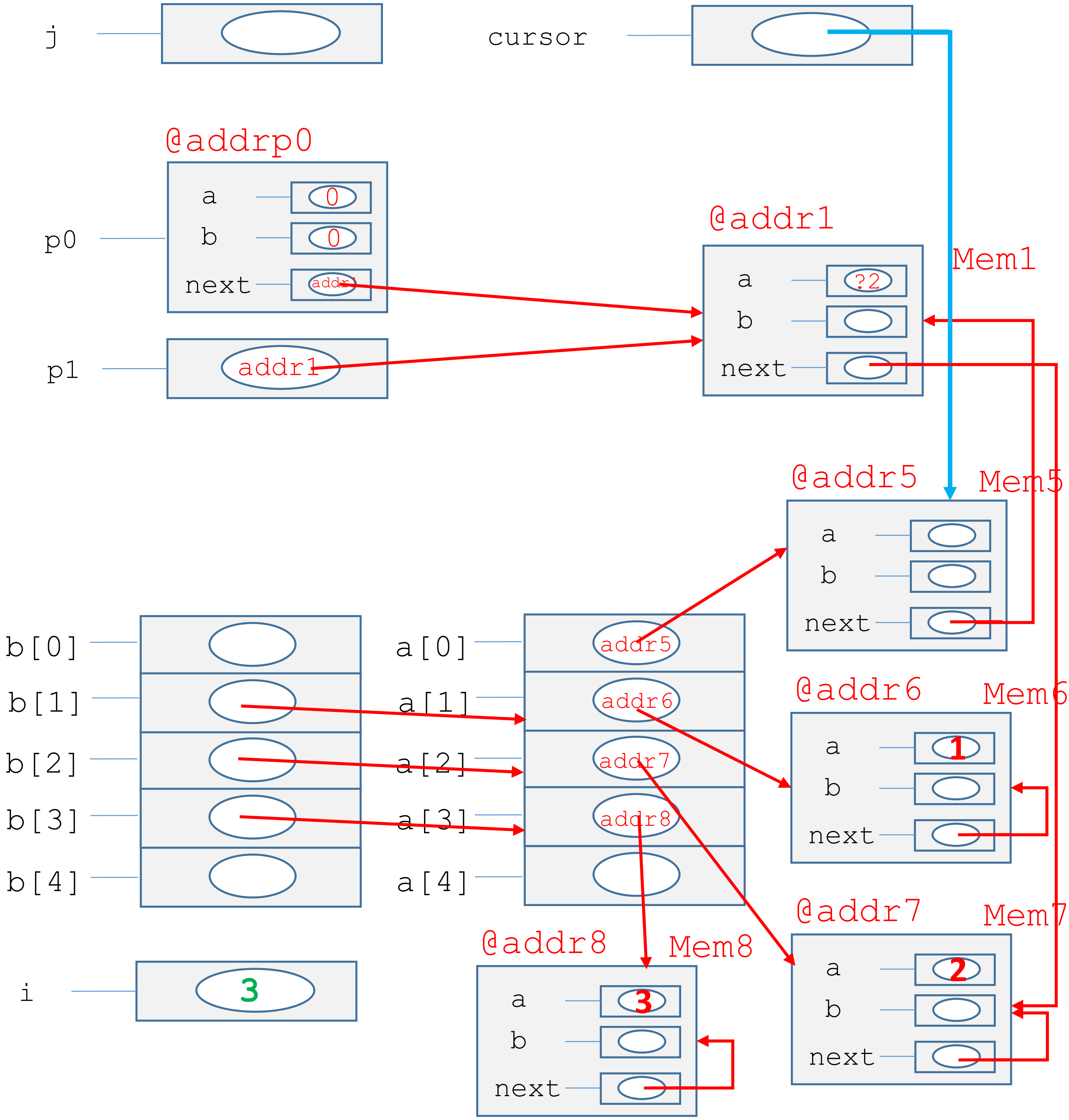
```
cursor = a[0];
for (j = 0; j < 5; j++)
{
          cursor = (*cursor).next;
}
```



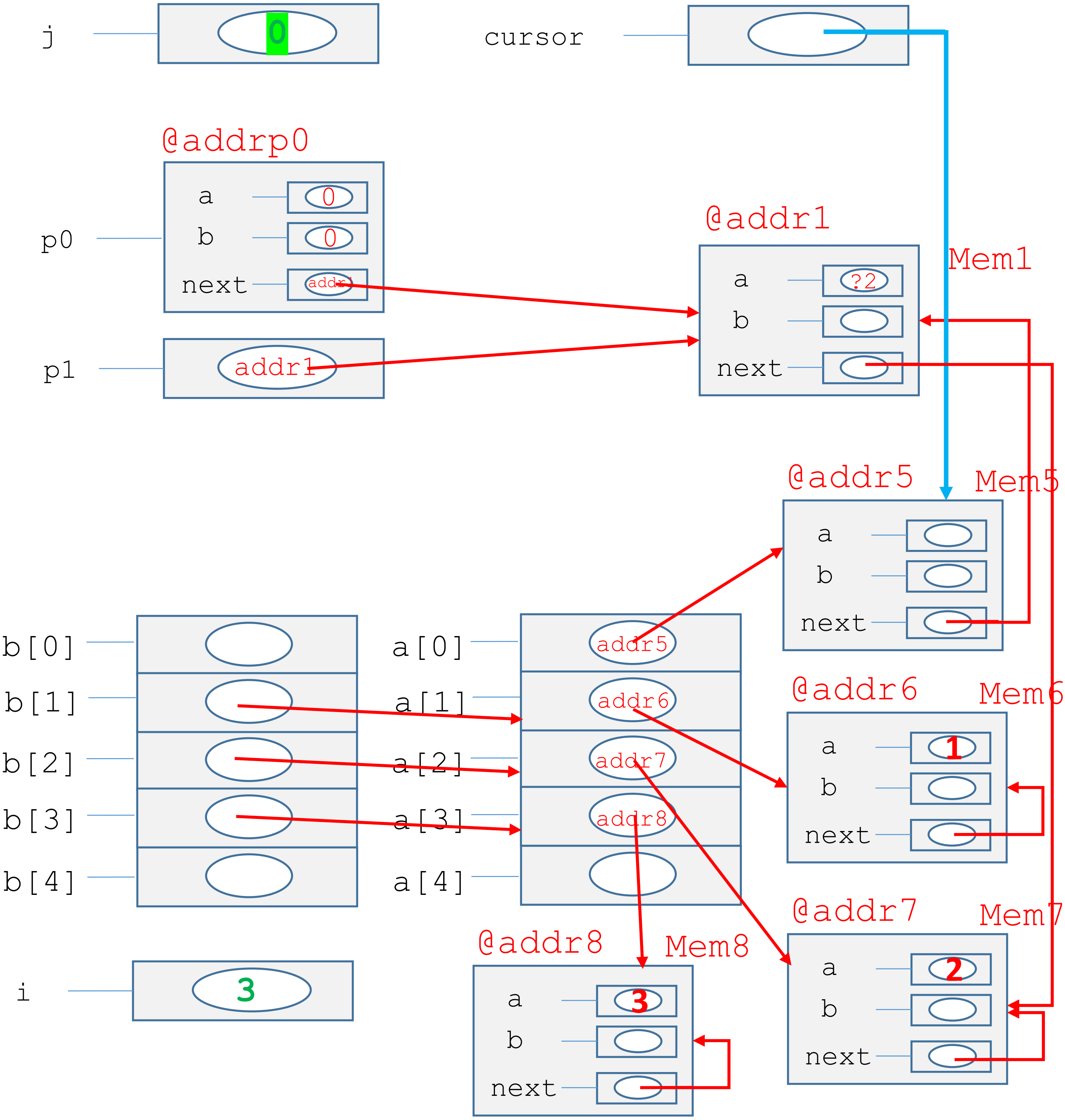
```
cursor = a[0];
for (j = 0; j < 5; j++)
{
        cursor = (*cursor).next;
}
```



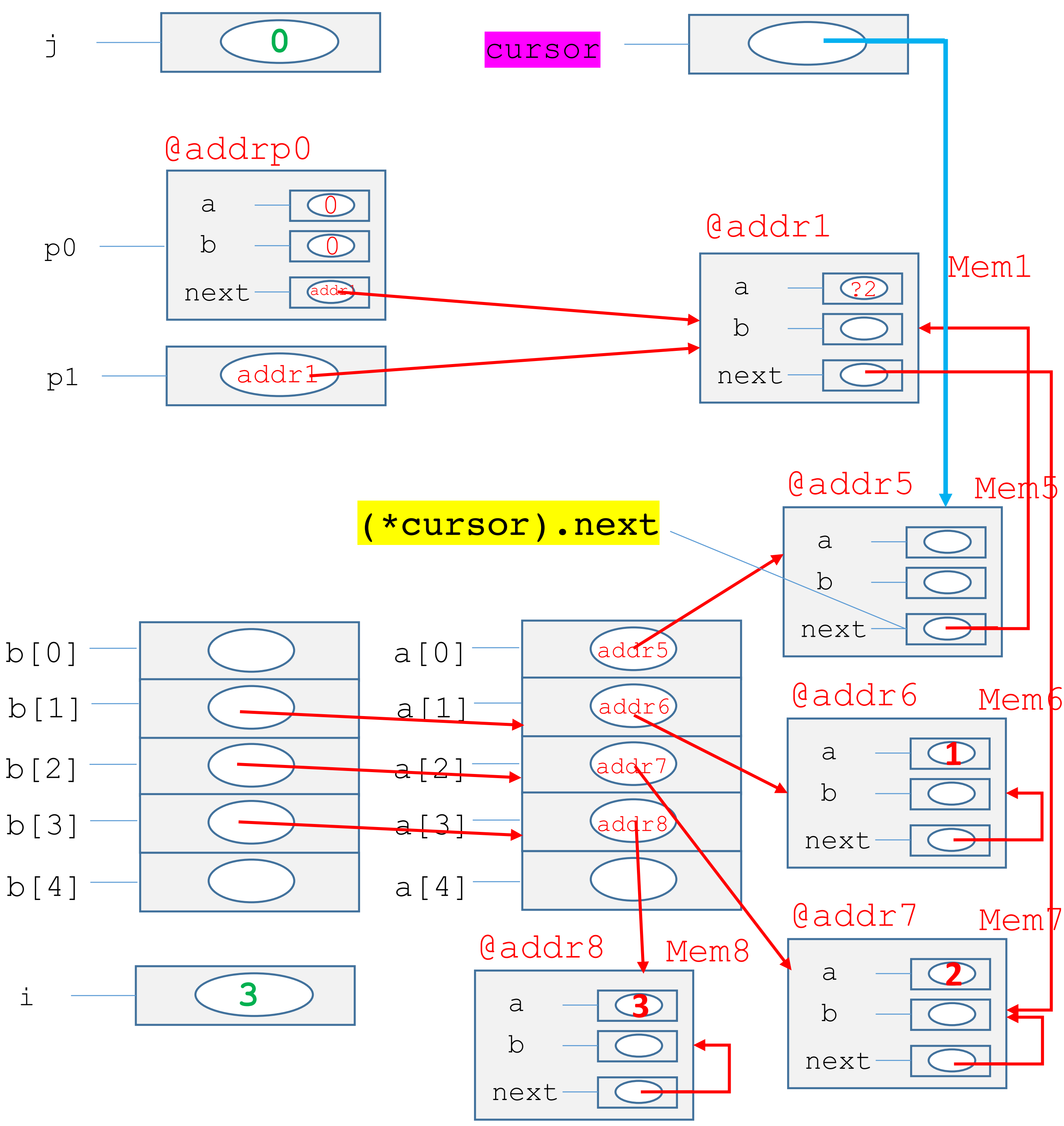
```
cursor = a[0];
for (j = 0; j < 5; j++)
{
        cursor = (*cursor).next;
}
```



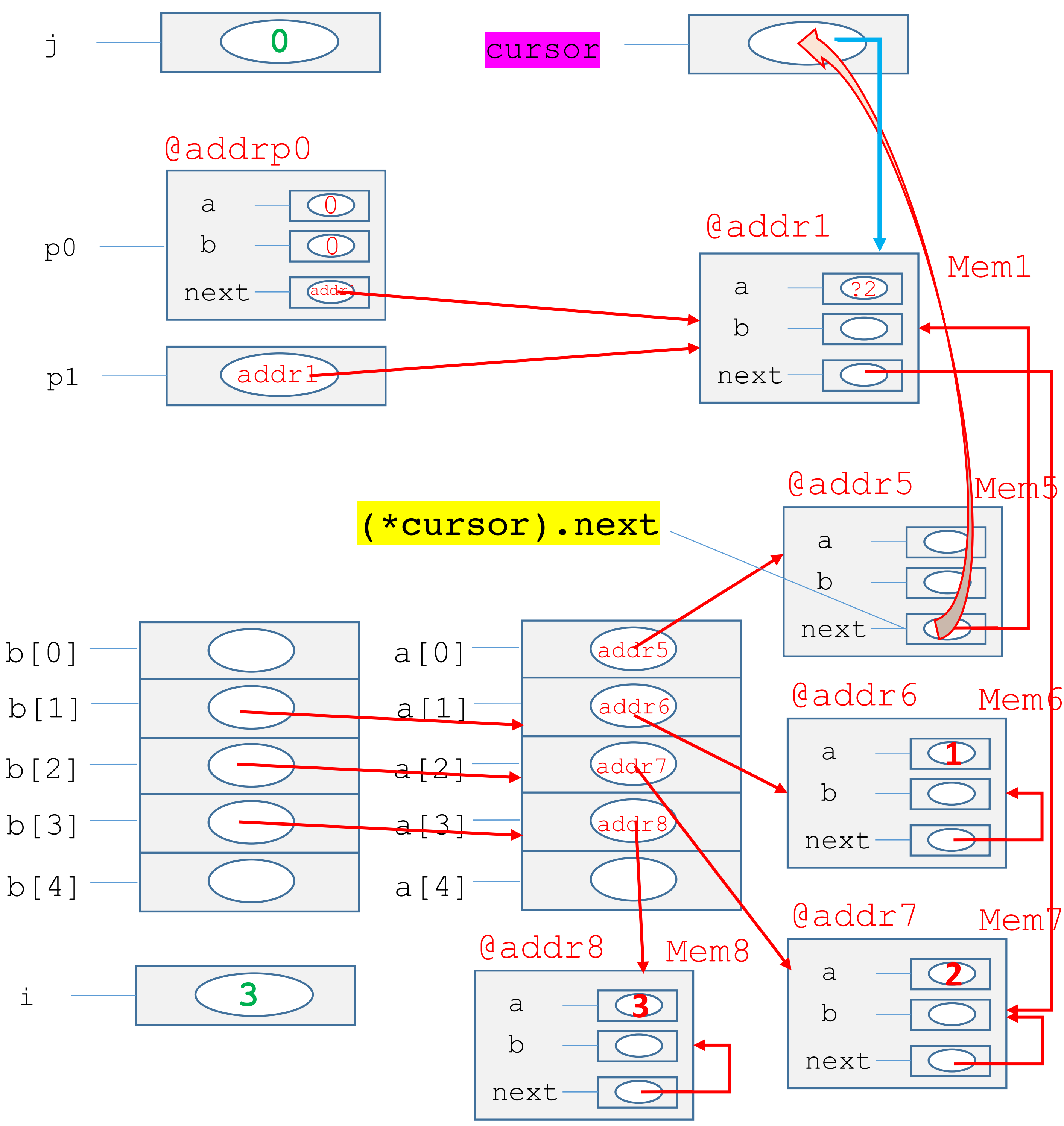
```
cursor = a[0];
for (j = 0; j < 5; j++)
{
        cursor = (*cursor).next;
}
```



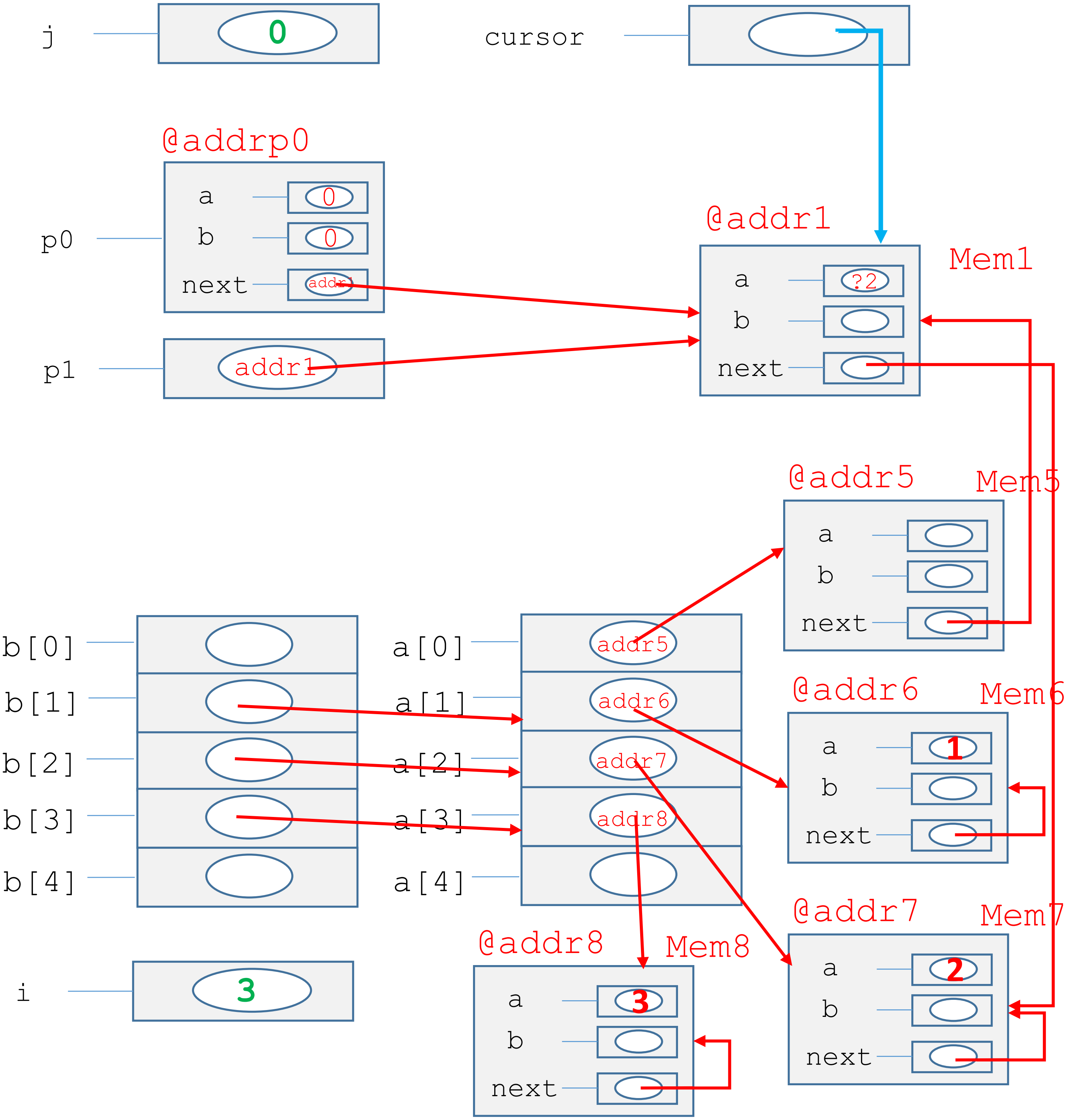
```
cursor = a[0];
for (j = 0; j < 5; j++)
{
        cursor = (*cursor).next;
}
```



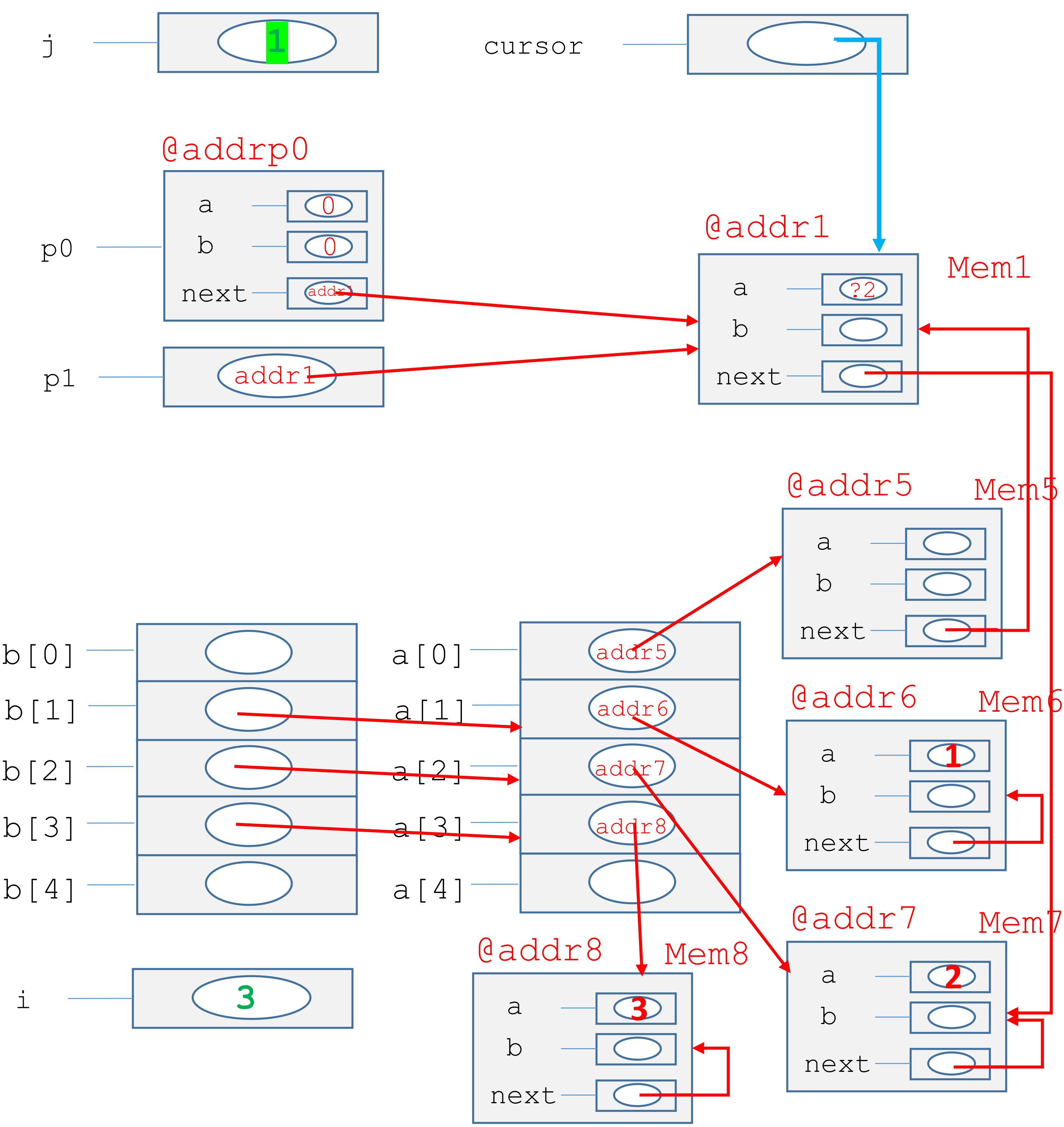
```
cursor = a[0];
for (j = 0; j < 5; j++)
{
        cursor = (*cursor).next;
}
```



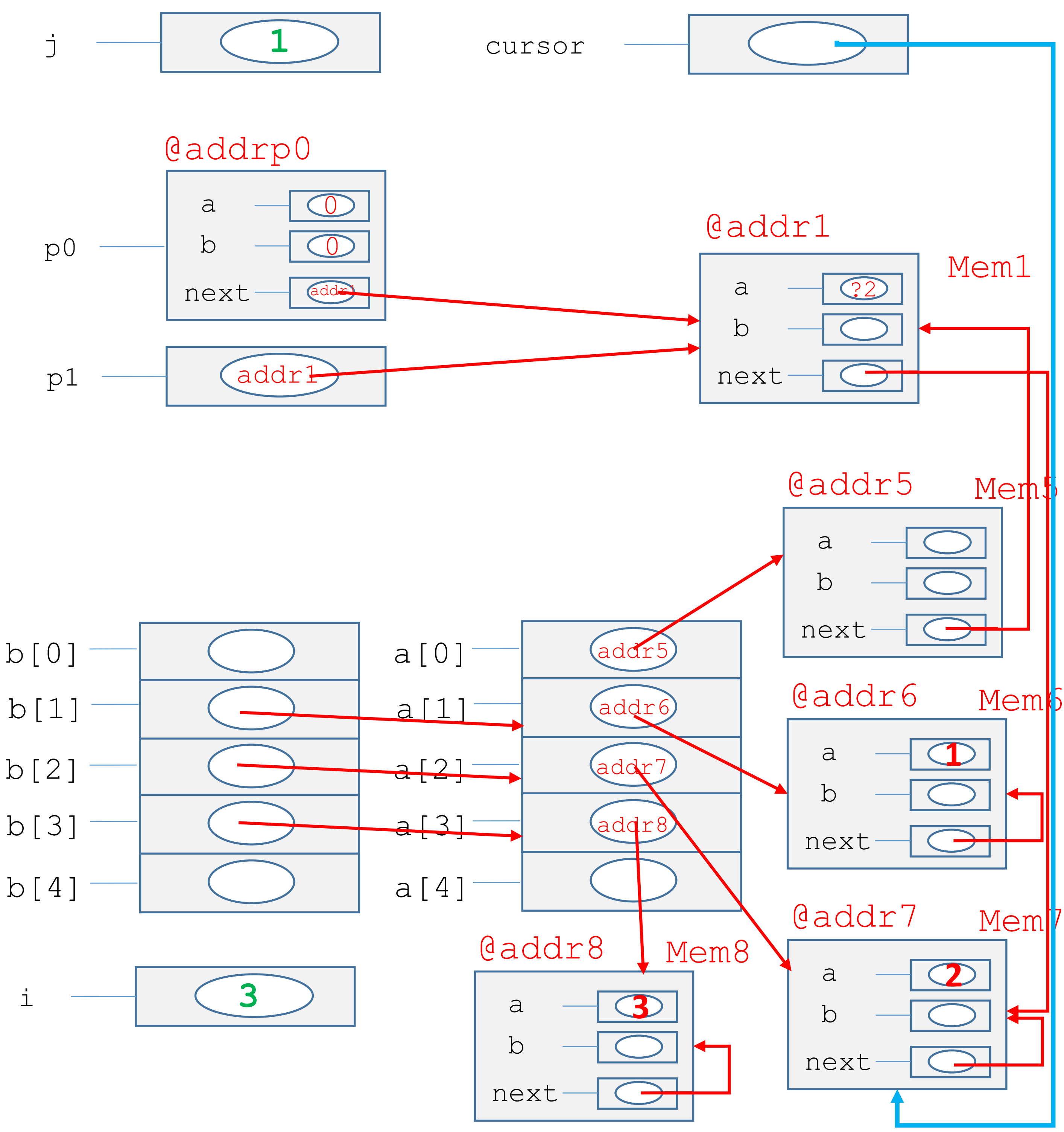
```
cursor = a[0];
for (j = 0; j < 5; j++)
{
          cursor = (*cursor).next;
}
```



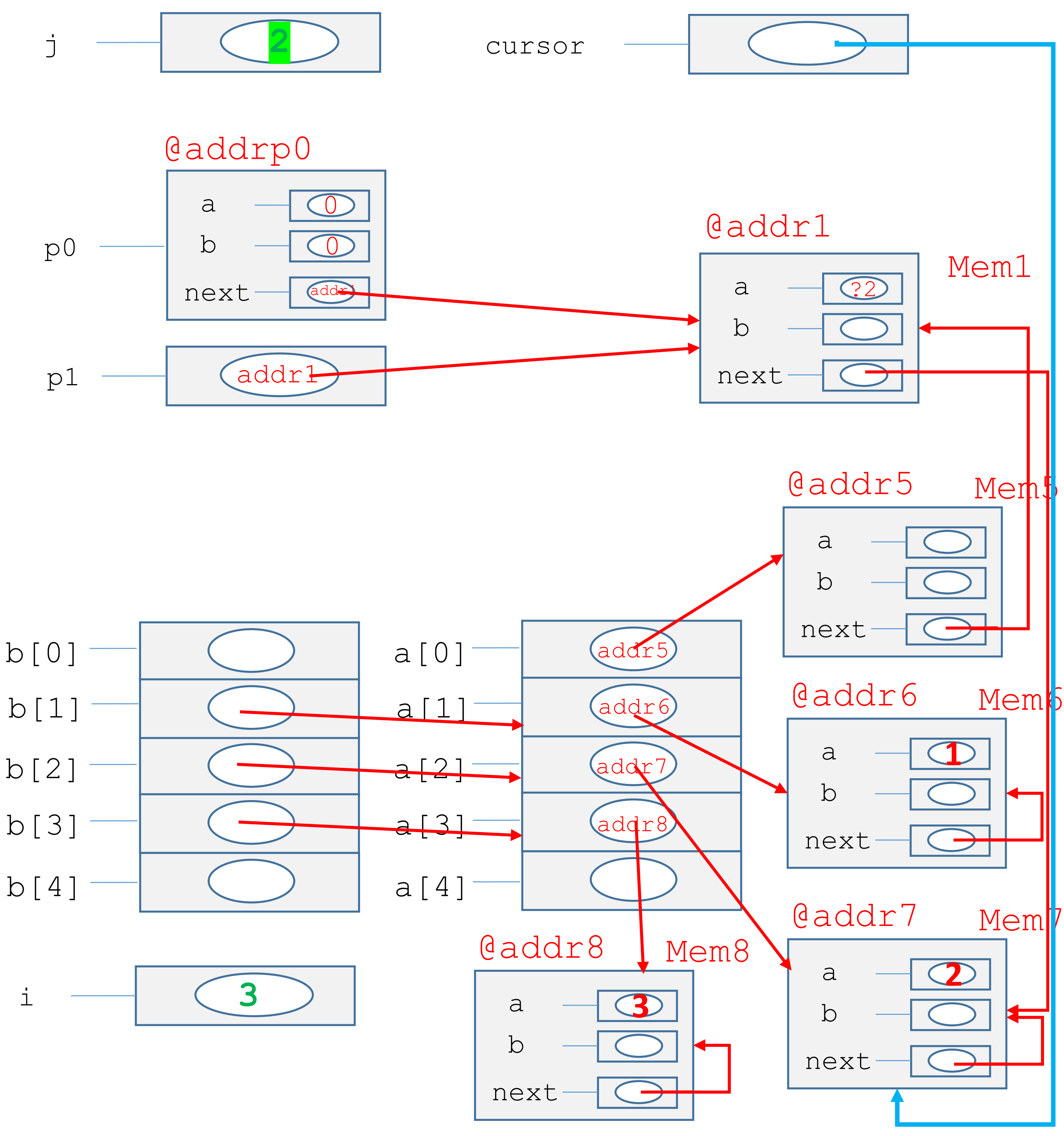
```
cursor = a[0];
for (j = 1; j < 5; j++)
{
        cursor = (*cursor).next;
}
```



```
cursor = a[0];
for (j = 1; j < 5; j++)
{
        cursor = (*cursor).next;
}
```

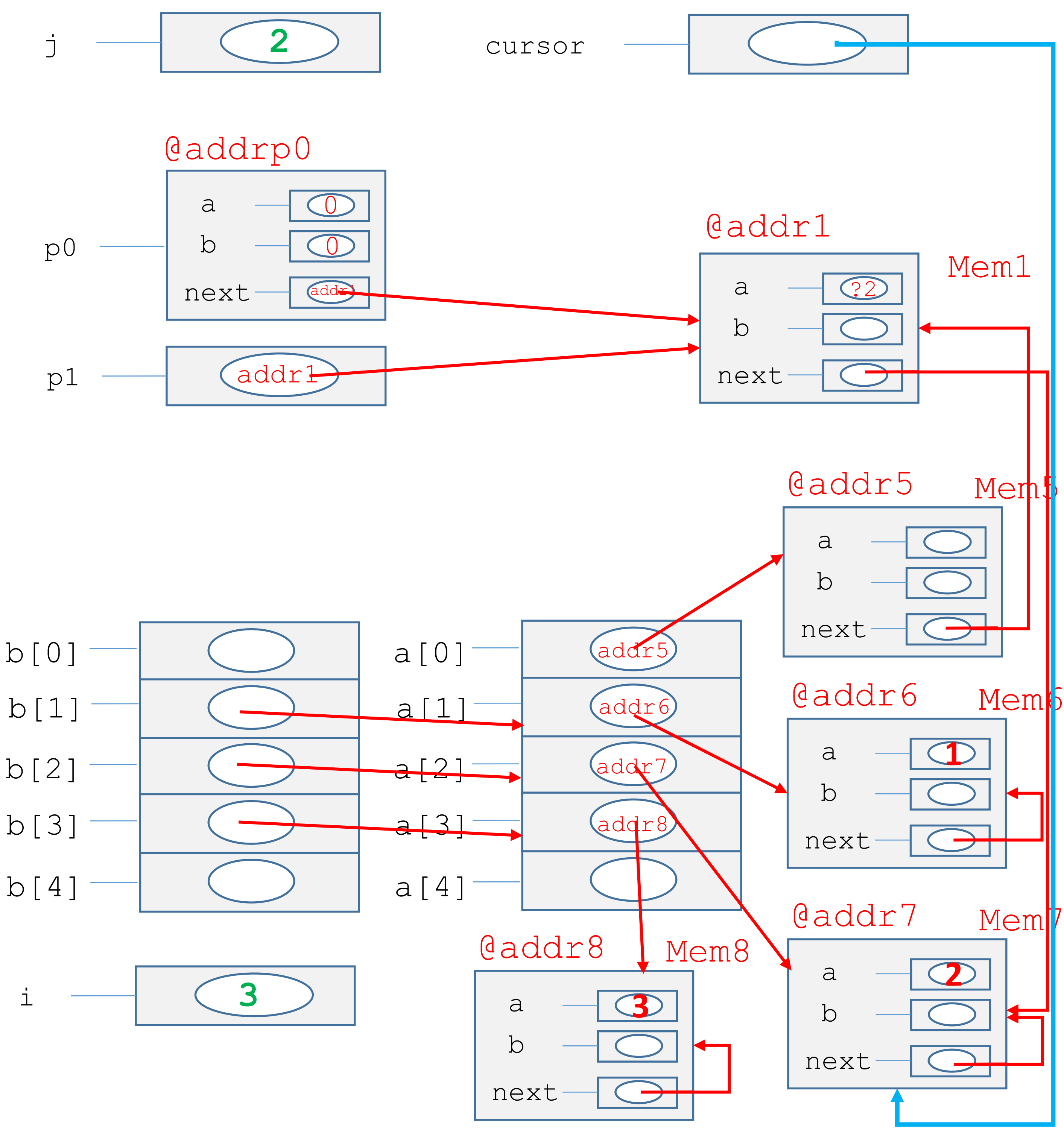


```
cursor = a[0];
for (j = 2; j < 5; j++)
{
        cursor = (*cursor).next;
}
```

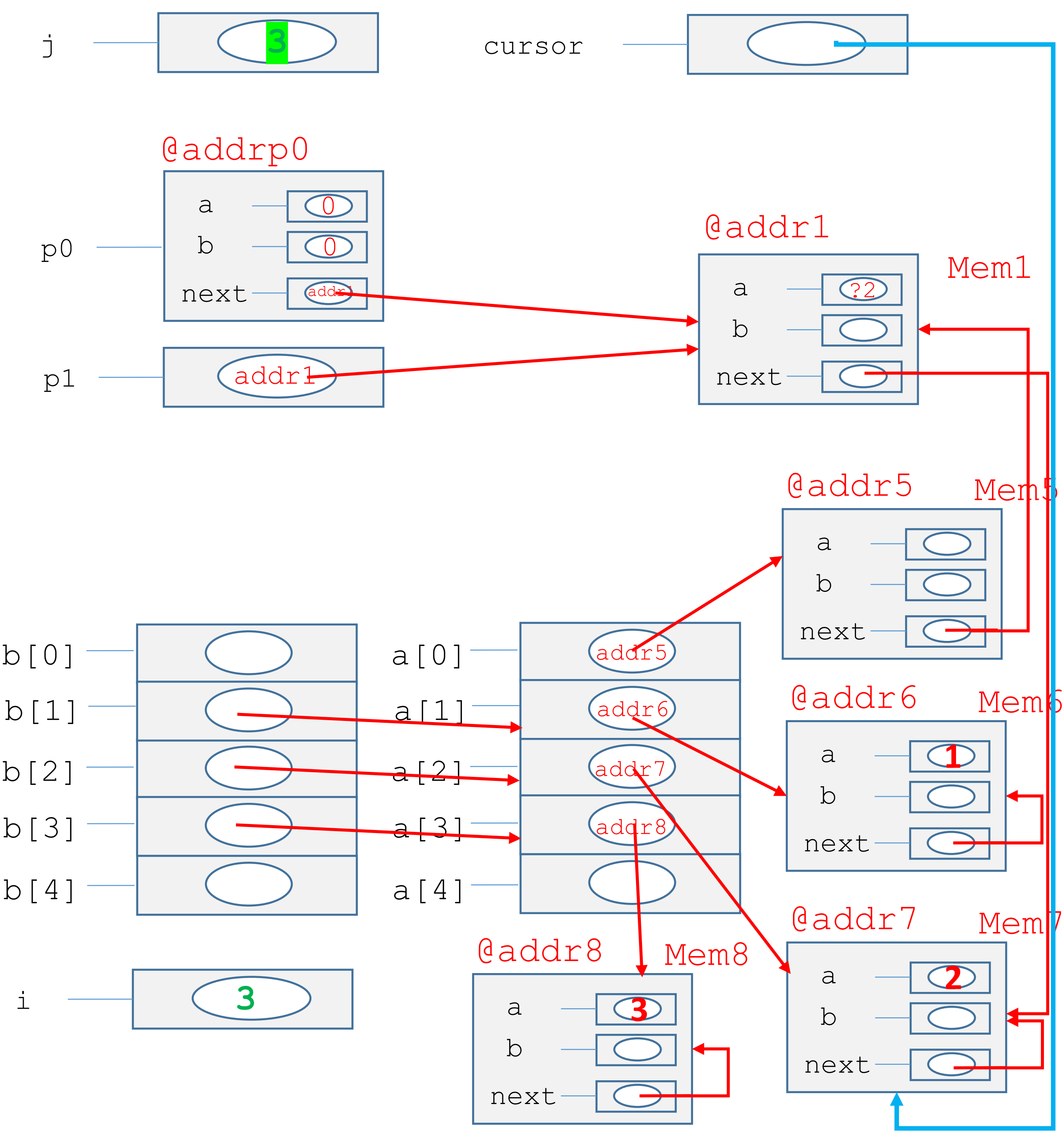


```
cursor = a[0];
for (j = 2; j < 5; j++)
{
        cursor = (*cursor).next;
}
```

cursor unchanged

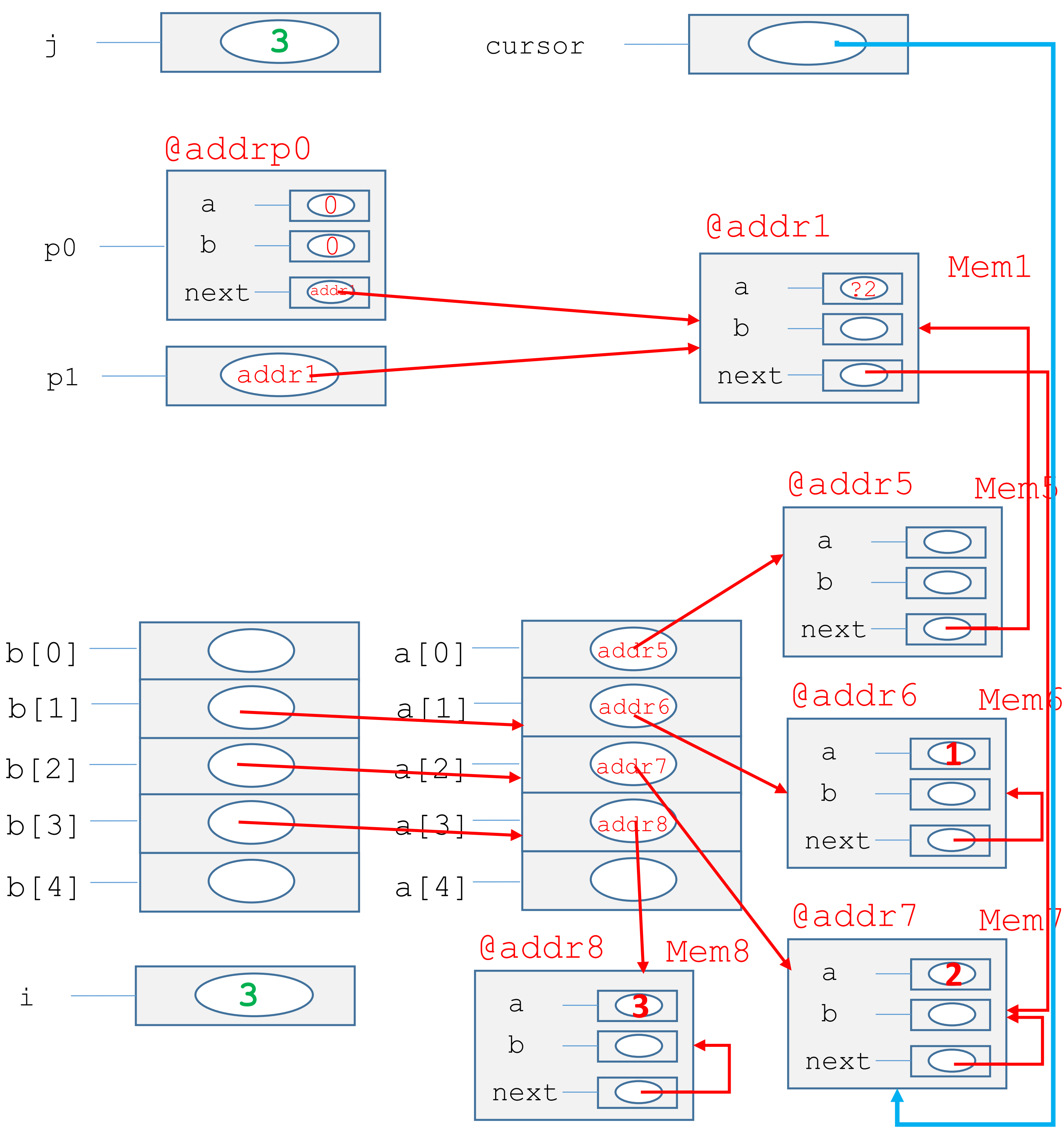


```
cursor = a[0];
for (j = 3; j < 5; j++)
{
        cursor = (*cursor).next;
}
```



```
cursor = a[0];
for (j = 3; j < 5; j++)
{
        cursor = (*cursor).next;
}
```

cursor unchanged


j
3
cursor
@addrp0
p0
a
0
b
0
next
addr1
p1
addr1
@addr1
Mem1
a
?2
b
next
@addr5
Mem5
a
b
next
@addr6
Mem6
a
1
b
next
@addr7
Mem7
a
2
b
next
@addr8
Mem8
a
3
b
next
b[0]
b[1]
b[2]
b[3]
b[4]
a[0]
addr5
a[1]
addr6
a[2]
addr7
a[3]
addr8
a[4]
i
3