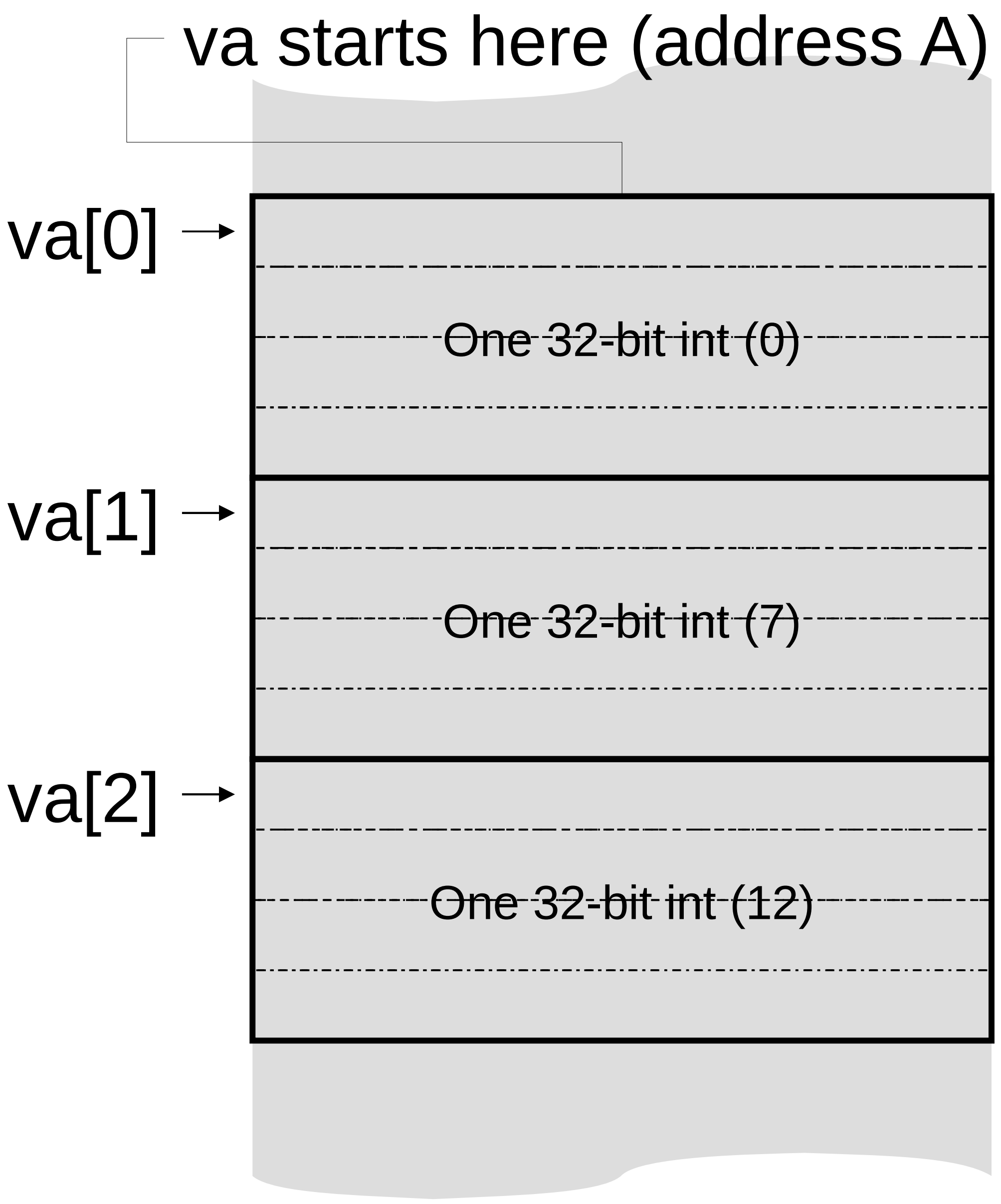
CS 101: Computer Programming and Utilization

10-Pointers

Instructor: Sridhar Iyer
IIT Bombay

Arrays and memory locations

- Elements of array have fixed size
- Laid out consecutively
- Compiler associates array name va with memory address of first byte
- To fetch va[ix], go to byte address A + ix*4 and fetch next four bytes as integer
- A, A+4, A+8, A+12, …



Pointers

Pointer: Object whose value represents the location of another object.

In C++ there are pointer types for each type of object:

Pointers to int objects; char objects; “UserDefined” objects

Even pointers to pointers.

Each pointer contains an address, which itself is a 32 bit number, representing an address from 0 to 2^{32} -1.

Why do we need pointers? - (we will see in later classes) parameter passing, dynamic memory allocation, so on.

Pointers basics

- A name defined as pointer type contains an address
- int m = 573;
- int* p; //This defines a name ‘p’ to have a type ‘int*’
  – Is a pointer to a value of type ‘int’
    - p cannot be assigned any value directly; It should be assigned a value through ‘address’ operator &
- p = &m;
    - &m: means address of m; //address operator
- cout << *p; //will print 573
    - *p: means contents of address stored in p; //dereferencing operator

Pointers basics (contd.)

An asterisk * has two uses with regard to pointers

In a definition, * indicates that the object is a pointer

```
int* p; int *q; int * r; //p,q,r are of type pointer to int
```

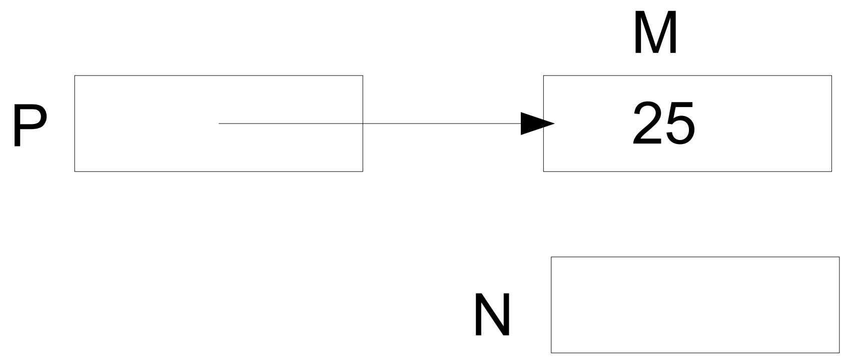
In expressions, when applied to a pointer, * evaluates to the object to which the pointer points

```
int m = 573;
p = &m; // p points to m
cout << *p;  // displays 573
```

Pointers visualization

- int m, n; int* p;
  m = 25
  p = &m;
  n = *p + 3;

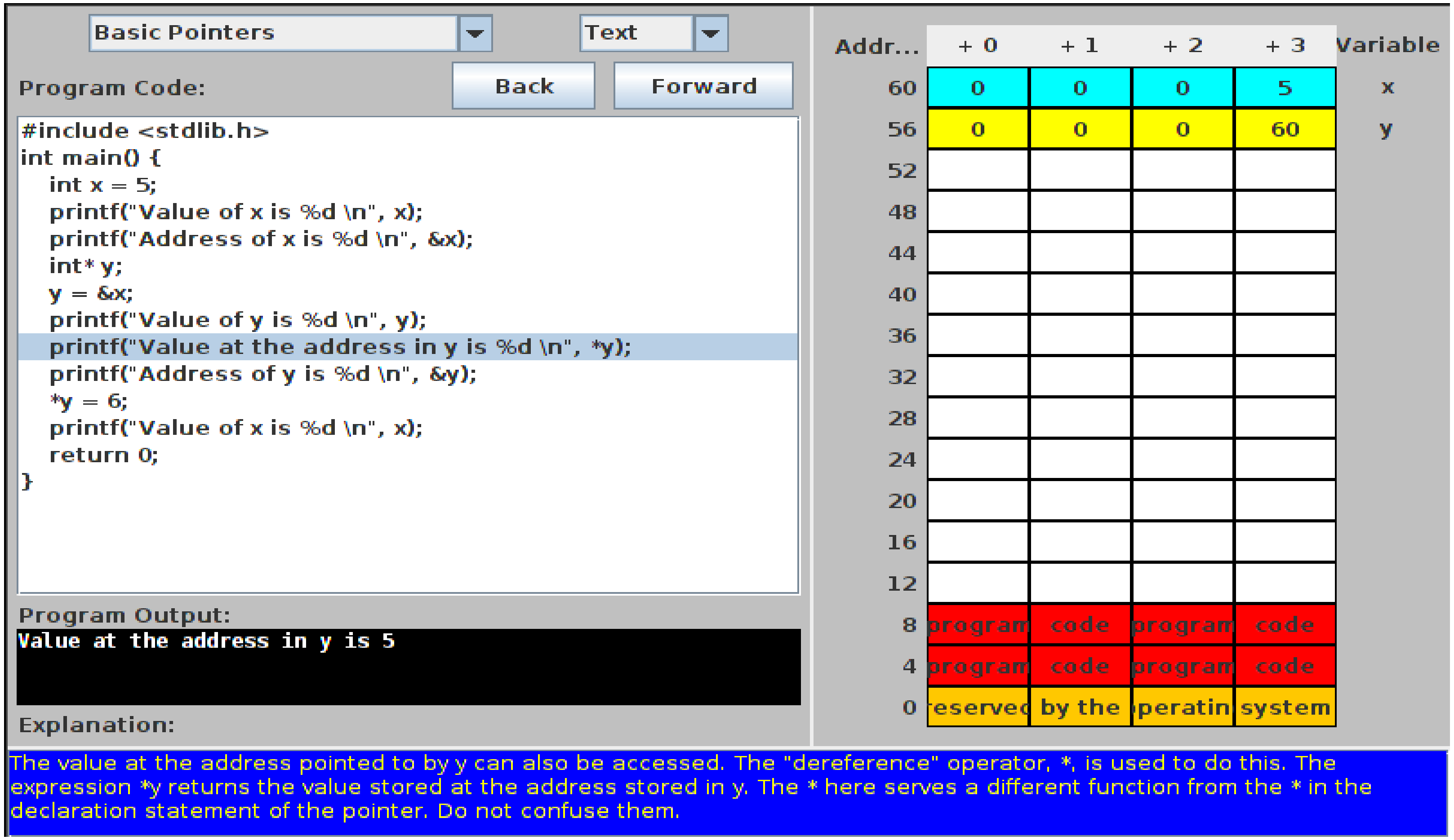
- What will be the value of n?



Fun: Watch Binky Pointer video

Pointers applet visualization

- Demos of code + memory visualization
  - http://www2.sis.pitt.edu/~is2470pb/Fall02/Final/rm/interactive.html
  - View basic pointers, click through quickly



```
#include <stdlib.h>
int main() {
    int x = 5;
    printf("Value of x is %d \n", x);
    printf("Address of x is %d \n", &x);
    int* y;
    y = &x;
    printf("Value of y is %d \n", y);
    printf("Value at the address in y is %d \n", *y);
    printf("Address of y is %d \n", &y);
    *y = 6;
    printf("Value of x is %d \n", x);
    return 0;
}
```

Program Output:

```
Value at the address in y is 5
```

Explanation:

The value at the address pointed to by y can also be accessed. The "dereference" operator, *, is used to do this. The expression *y returns the value stored at the address stored in y. The * here serves a different function from the * in the declaration statement of the pointer. Do not confuse them.

Addr...	+ 0	+ 1	+ 2	+ 3	Variable
60	0	0	0	5	x
56	0	0	0	60	y
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	eserved	by the	peratin	system	

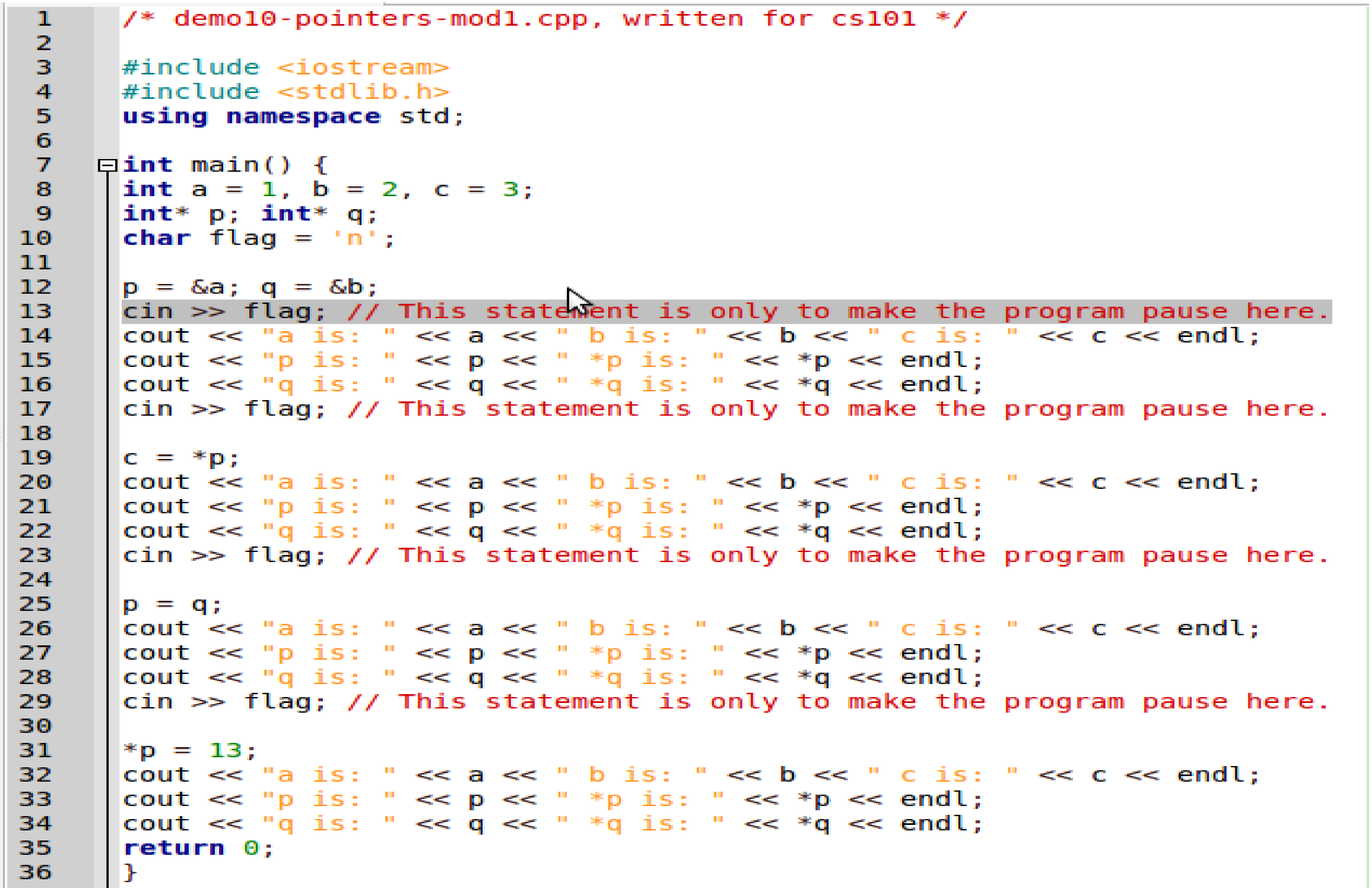
Predict the output of the program below

- int a = 1, b = 2, c = 3;
- int* p, int* q;

```
p = &a; q = &b;
c = *p; p = q;
*p = 13;
cout << a << b << c << endl;
cout << *p << *q << endl;
```

- What values will be printed?
  - Hint: Draw boxes and arrows as in previous slides
  - Run: demo10-pointers.cpp

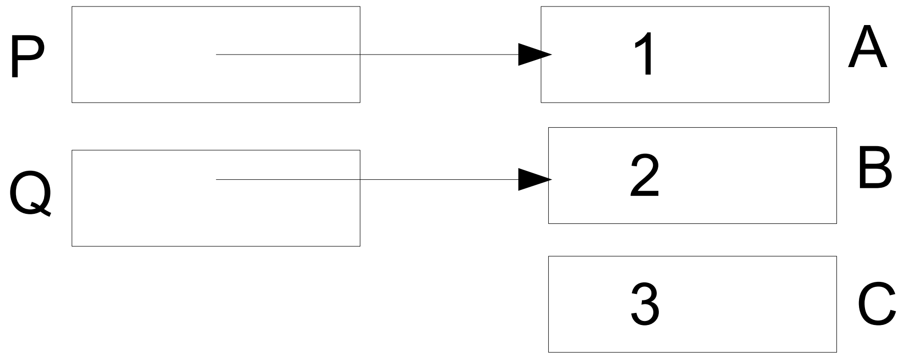
demo10-pointers.cpp – predict output 1

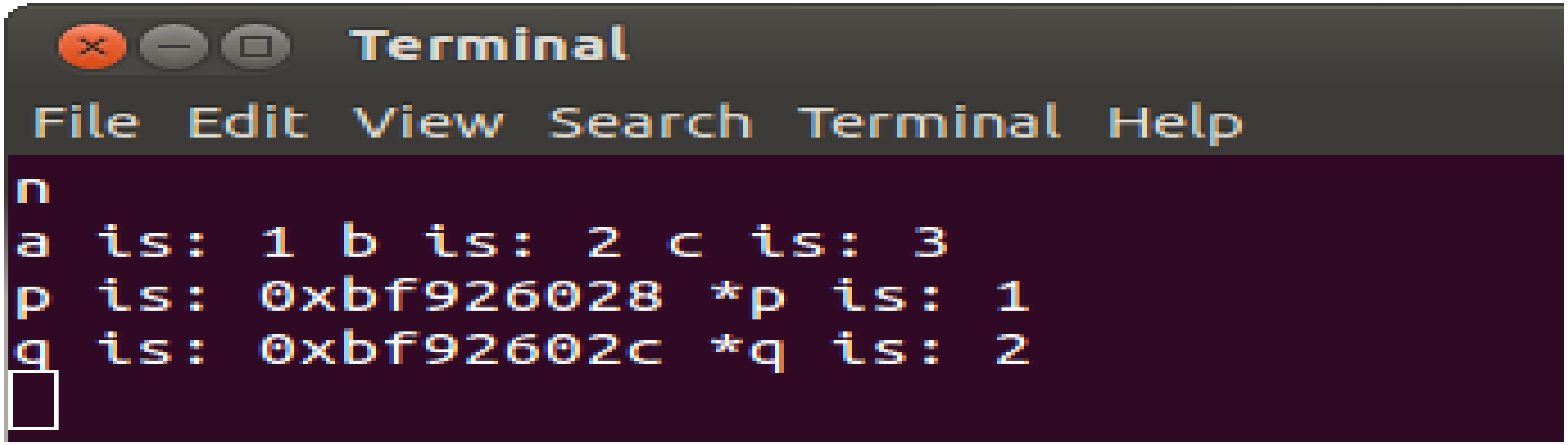


```cpp
/* demo10-pointers-mod1.cpp, written for cs101 */

#include <iostream>
#include <stdlib.h>
using namespace std;

int main() {
int a = 1, b = 2, c = 3;
int* p; int* q;
char flag = 'n';

p = &a; q = &b;
cin >> flag; // This statement is only to make the program pause here.
cout << "a is: " << a << " b is: " << b << " c is: " << c << endl;
cout << "p is: " << p << " *p is: " << *p << endl;
cout << "q is: " << q << " *q is: " << *q << endl;
cin >> flag; // This statement is only to make the program pause here.

c = *p;
cout << "a is: " << a << " b is: " << b << " c is: " << c << endl;
cout << "p is: " << p << " *p is: " << *p << endl;
cout << "q is: " << q << " *q is: " << *q << endl;
cin >> flag; // This statement is only to make the program pause here.

p = q;
cout << "a is: " << a << " b is: " << b << " c is: " << c << endl;
cout << "p is: " << p << " *p is: " << *p << endl;
cout << "q is: " << q << " *q is: " << *q << endl;
cin >> flag; // This statement is only to make the program pause here.

*p = 13;
cout << "a is: " << a << " b is: " << b << " c is: " << c << endl;
cout << "p is: " << p << " *p is: " << *p << endl;
cout << "q is: " << q << " *q is: " << *q << endl;
return 0;
}
```

Check your answer; predict output 2

```
int main() {
int a = 1, b = 2, c = 3;
int* p; int* q;
char flag = 'n';

p = &a; q = &b;
```

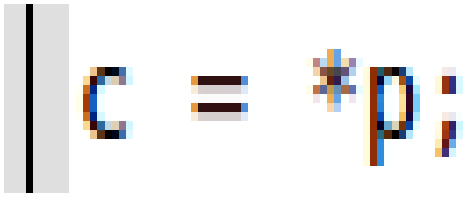




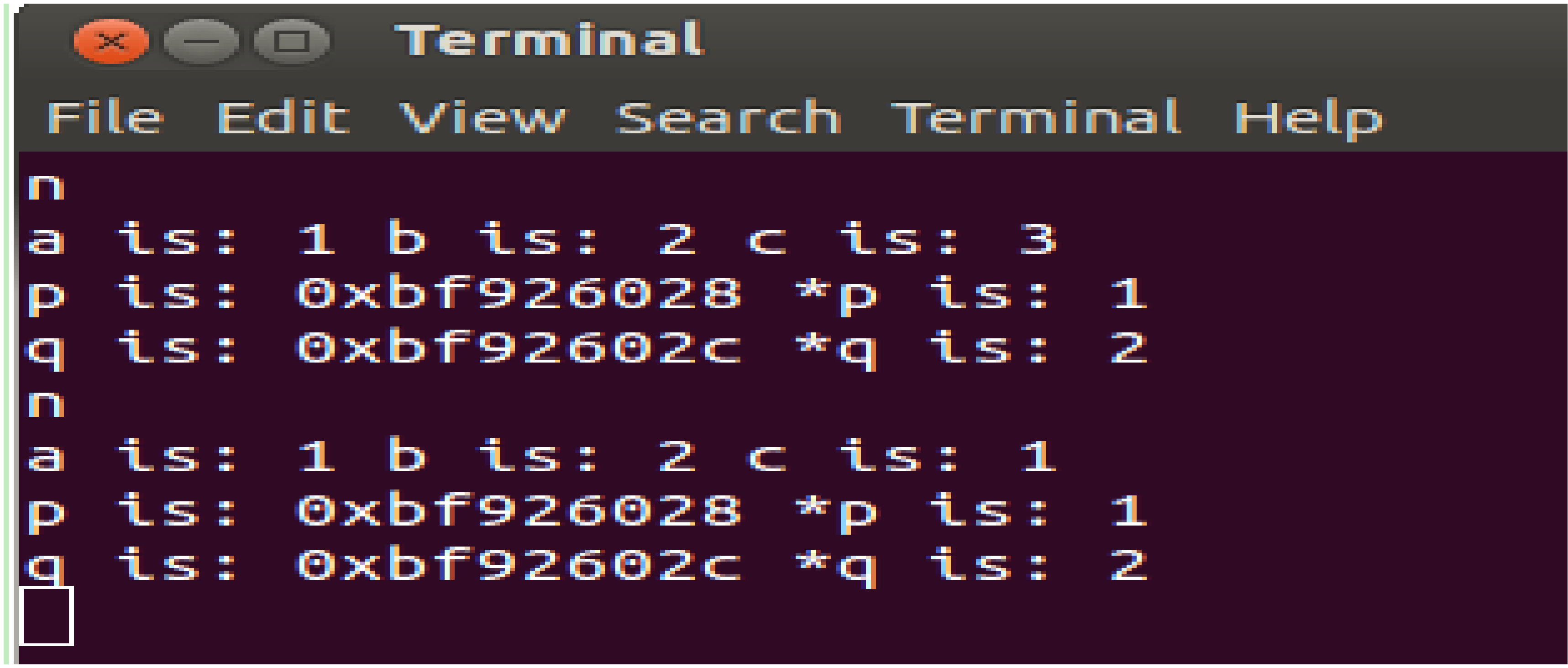
```
n
a is: 1 b is: 2 c is: 3
p is: 0xbf926028 *p is: 1
q is: 0xbf92602c *q is: 2
```

```
c = *p;
```

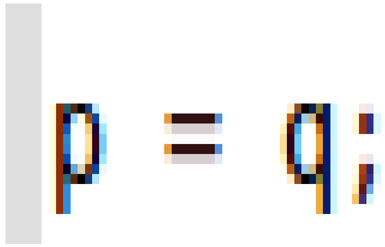
Check your answer; predict output 3



```
c = *p;
```



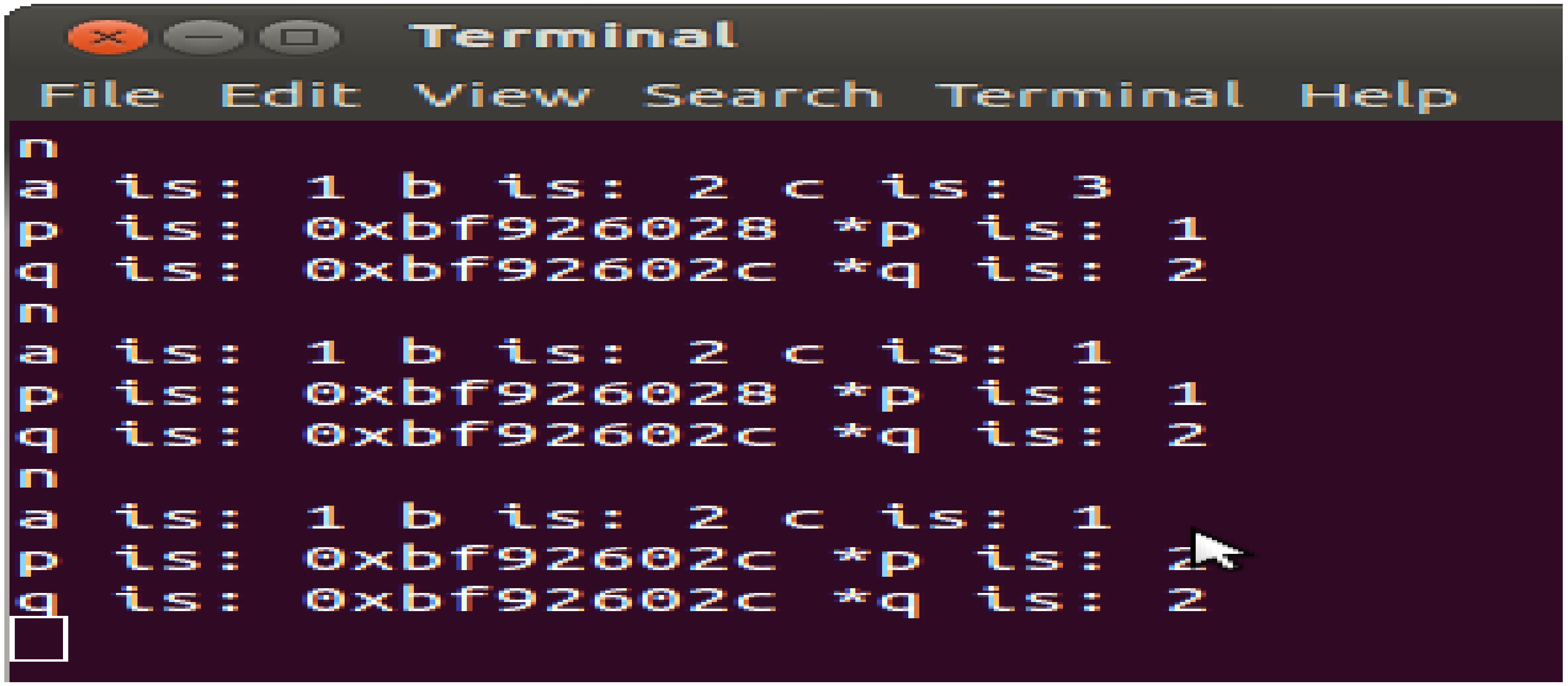
```
n
a is: 1 b is: 2 c is: 3
p is: 0xbf926028 *p is: 1
q is: 0xbf92602c *q is: 2
n
a is: 1 b is: 2 c is: 1
p is: 0xbf926028 *p is: 1
q is: 0xbf92602c *q is: 2
```



```
p = q;
```

Check your answer; predict output 4

```
p = q;
```

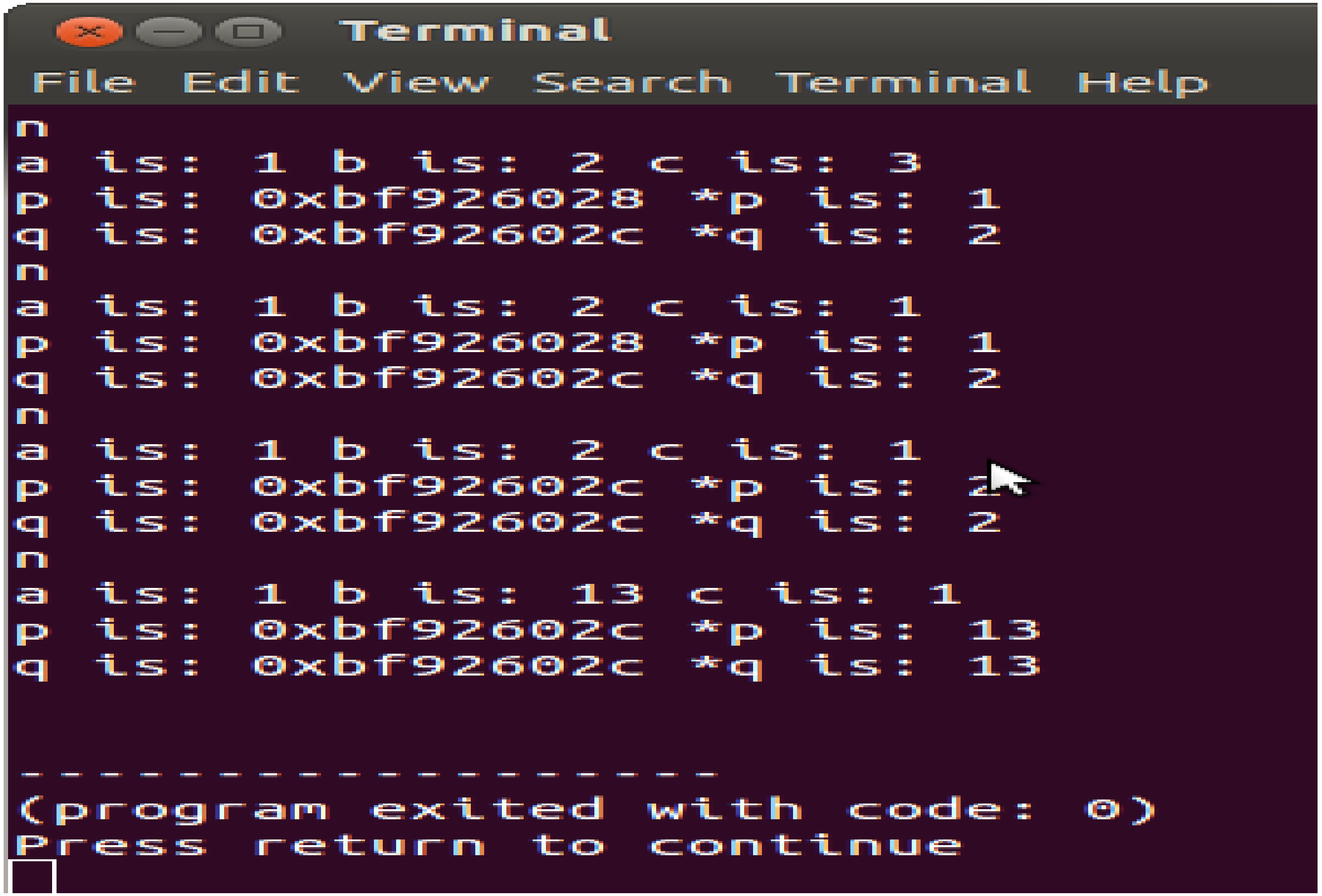


```
n
a is: 1 b is: 2 c is: 3
p is: 0xbf926028 *p is: 1
q is: 0xbf92602c *q is: 2
n
a is: 1 b is: 2 c is: 1
p is: 0xbf926028 *p is: 1
q is: 0xbf92602c *q is: 2
n
a is: 1 b is: 2 c is: 1
p is: 0xbf92602c *p is: 2
q is: 0xbf92602c *q is: 2
```

```
*p = 13;
```

Check your answer – end of program

```
*p = 13;
```

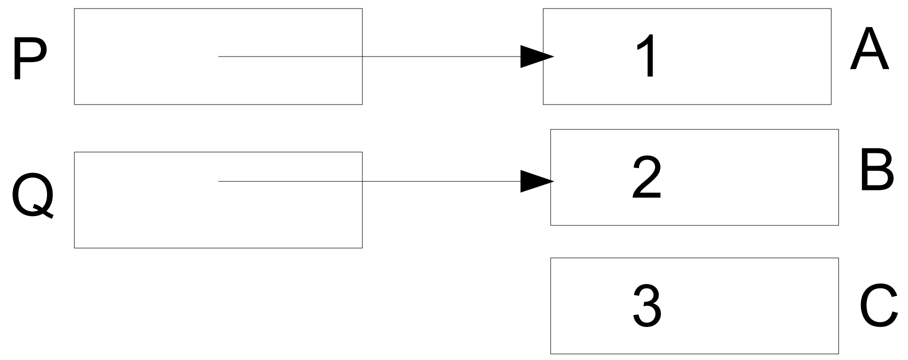


```
n
a is: 1 b is: 2 c is: 3
p is: 0xbf926028 *p is: 1
q is: 0xbf92602c *q is: 2
n
a is: 1 b is: 2 c is: 1
p is: 0xbf926028 *p is: 1
q is: 0xbf92602c *q is: 2
n
a is: 1 b is: 2 c is: 1
p is: 0xbf92602c *p is: 2
q is: 0xbf92602c *q is: 2
n
a is: 1 b is: 13 c is: 1
p is: 0xbf92602c *p is: 13
q is: 0xbf92602c *q is: 13


------------------
(program exited with code: 0)
Press return to continue
```

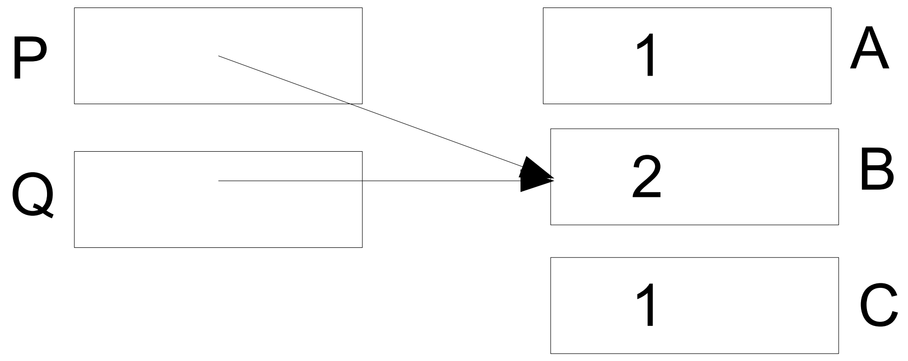
```
return 0;
```

Check your answers - visually

p = &a; q = &b;



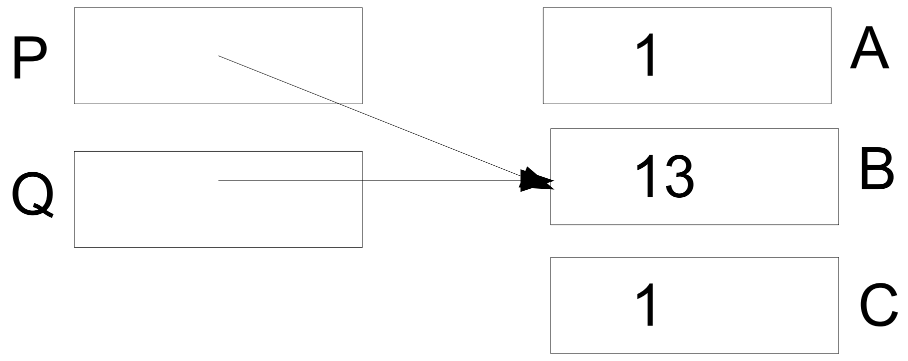
c = *p; p = q;



*p = 13;

cout: 1, 13, 1

cout: 13, 13

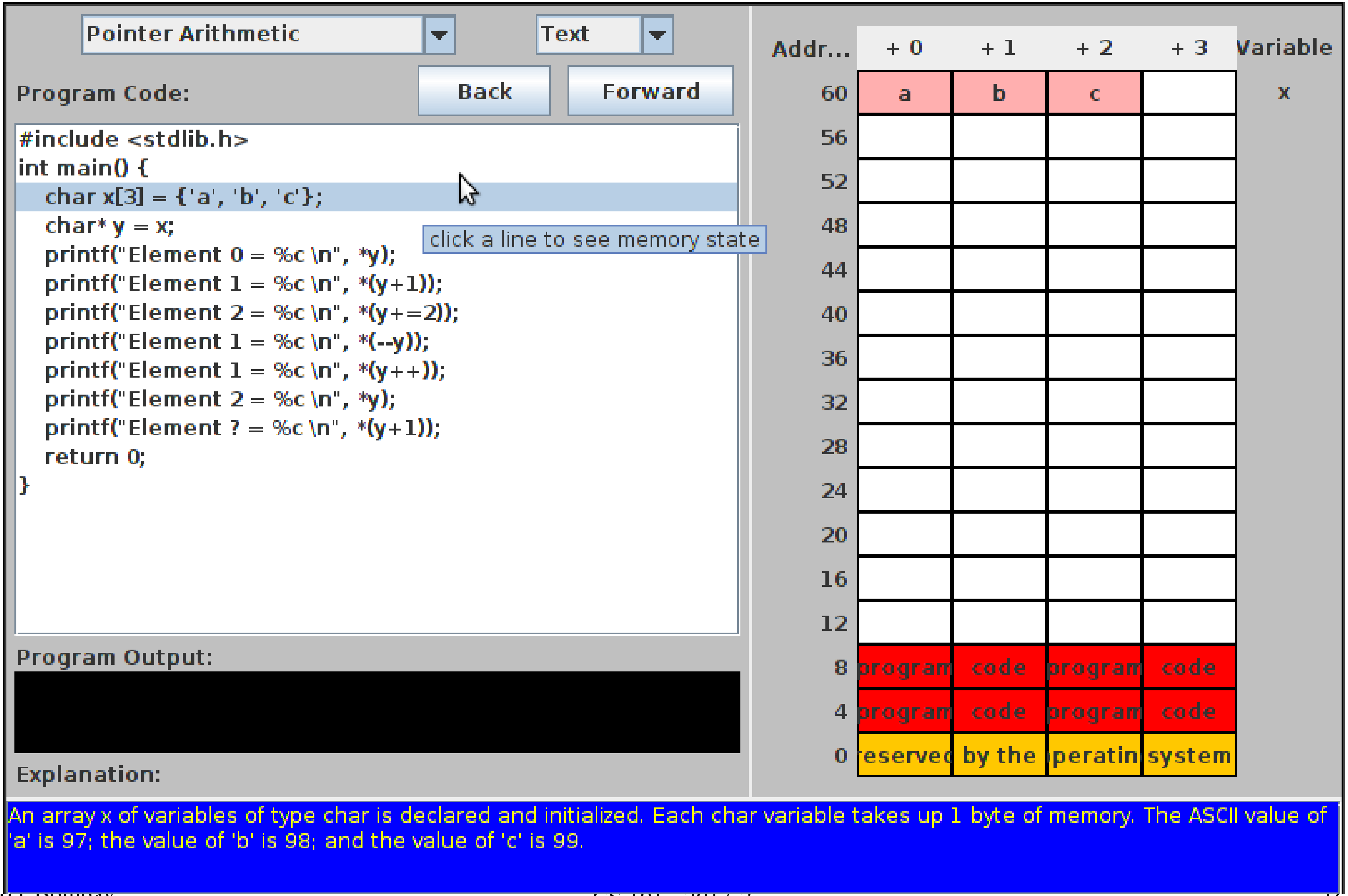


Pointer Arithmetic

- We can print the value of a pointer itself
  - The address stored there will be printed
    - cout << p1 // may print 0xbfde41a0

- Can also 'increment' the value of a pointer
    - p2 = &A[0]; p2++;
    - This will increment the contents of p2 by the no of bytes (4) allocated to the associated 'type' for array A, and not by 1
    - cout << *p2 // will now print value of A[1]
- Note: In case of arrays, p2 = A; is same as p2 = &A[0];

Predict what will happen at step 2



Program Code:

```
#include <stdlib.h>
int main() {
   char x[3] = {'a', 'b', 'c'};
   char* y = x;
   printf("Element 0 = %c \n", *y);
   printf("Element 1 = %c \n", *(y+1));
   printf("Element 2 = %c \n", *(y+=2));
   printf("Element 1 = %c \n", *(--y));
   printf("Element 1 = %c \n", *(y++));
   printf("Element 2 = %c \n", *y);
   printf("Element ? = %c \n", *(y+1));
   return 0;
}
```

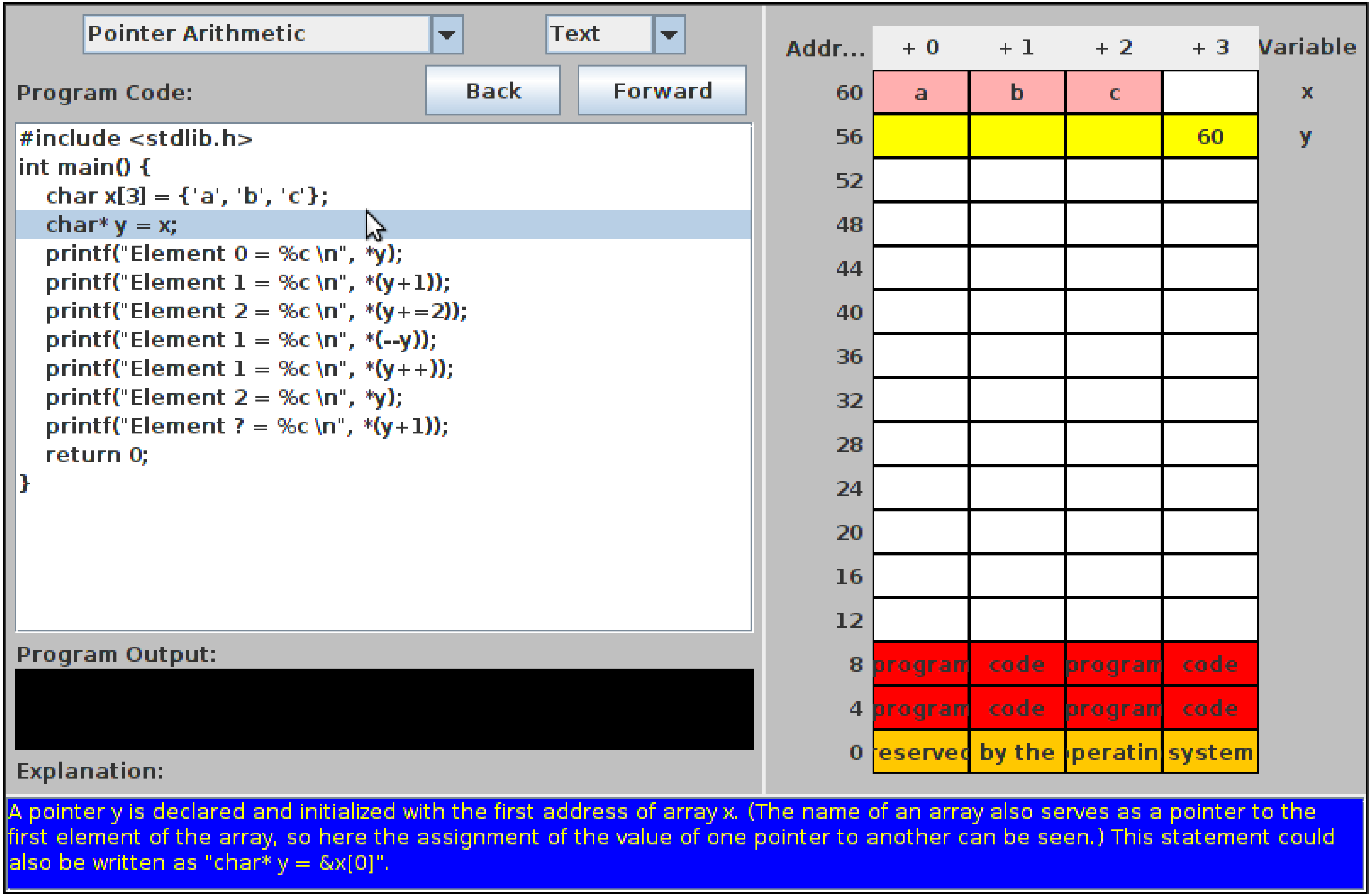
Program Output:

Addr...	+ 0	+ 1	+ 2	+ 3	Variable
60	a	b	c		x
56					
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	reserved	by the	operatin	system	

Explanation:

An array x of variables of type char is declared and initialized. Each char variable takes up 1 byte of memory. The ASCII value of 'a' is 97; the value of 'b' is 98; and the value of 'c' is 99.

Check your answer; predict steps 3&4

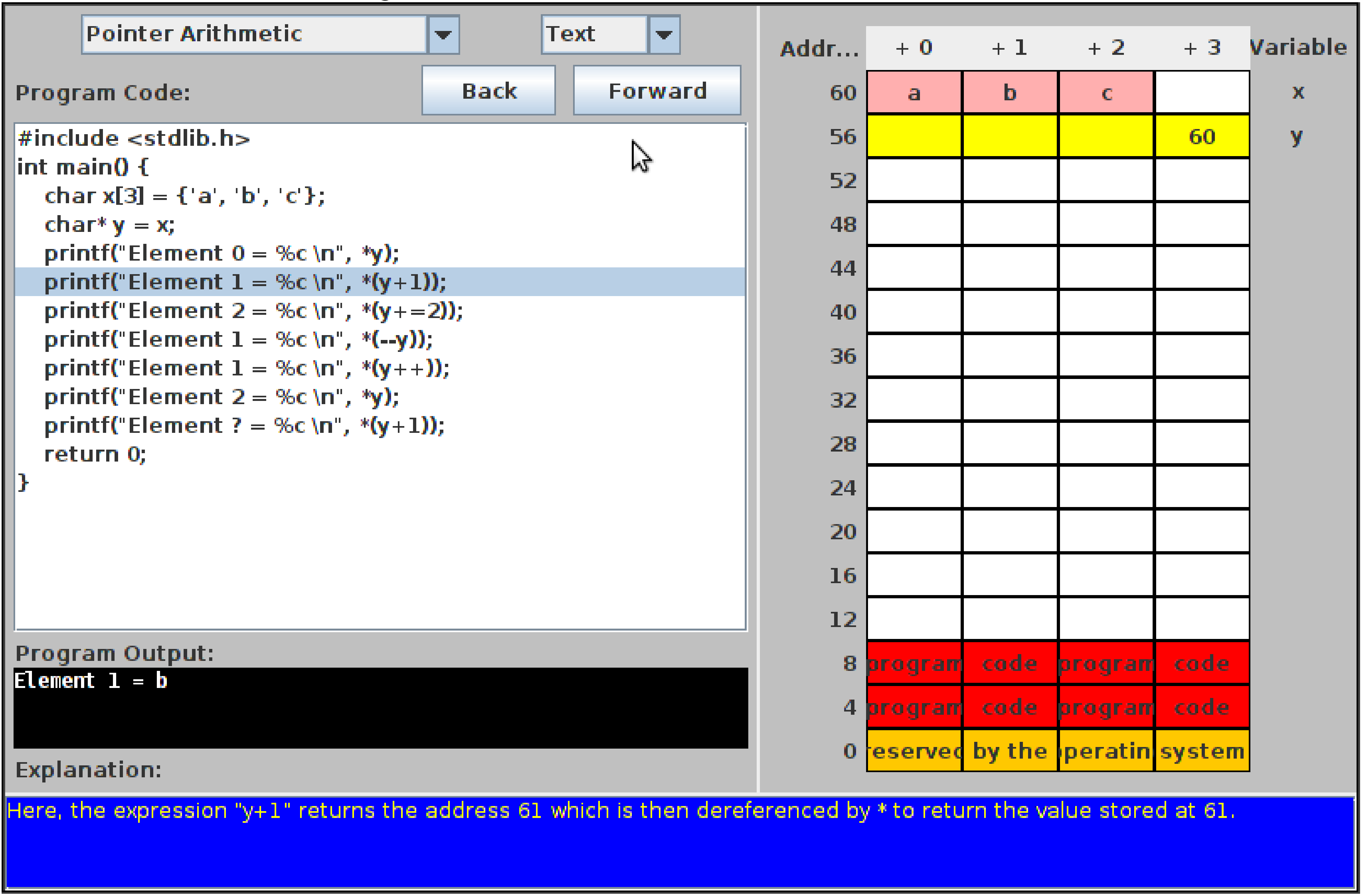


```
#include <stdlib.h>
int main() {
   char x[3] = {'a', 'b', 'c'};
   char* y = x;
   printf("Element 0 = %c \n", *y);
   printf("Element 1 = %c \n", *(y+1));
   printf("Element 2 = %c \n", *(y+=2));
   printf("Element 1 = %c \n", *(--y));
   printf("Element 1 = %c \n", *(y++));
   printf("Element 2 = %c \n", *y);
   printf("Element ? = %c \n", *(y+1));
   return 0;
}
```

Addr...	+ 0	+ 1	+ 2	+ 3	Variable
60	a	b	c		x
56				60	y
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	reserved	by the	operating	system	

A pointer y is declared and initialized with the first address of array x. (The name of an array also serves as a pointer to the first element of the array, so here the assignment of the value of one pointer to another can be seen.) This statement could also be written as "char* y = &x[0]".

Check your answer; predict step 5



Program Code:

```
#include <stdlib.h>
int main() {
   char x[3] = {'a', 'b', 'c'};
   char* y = x;
   printf("Element 0 = %c \n", *y);
   printf("Element 1 = %c \n", *(y+1));
   printf("Element 2 = %c \n", *(y+=2));
   printf("Element 1 = %c \n", *(--y));
   printf("Element 1 = %c \n", *(y++));
   printf("Element 2 = %c \n", *y);
   printf("Element ? = %c \n", *(y+1));
   return 0;
}
```

Program Output:

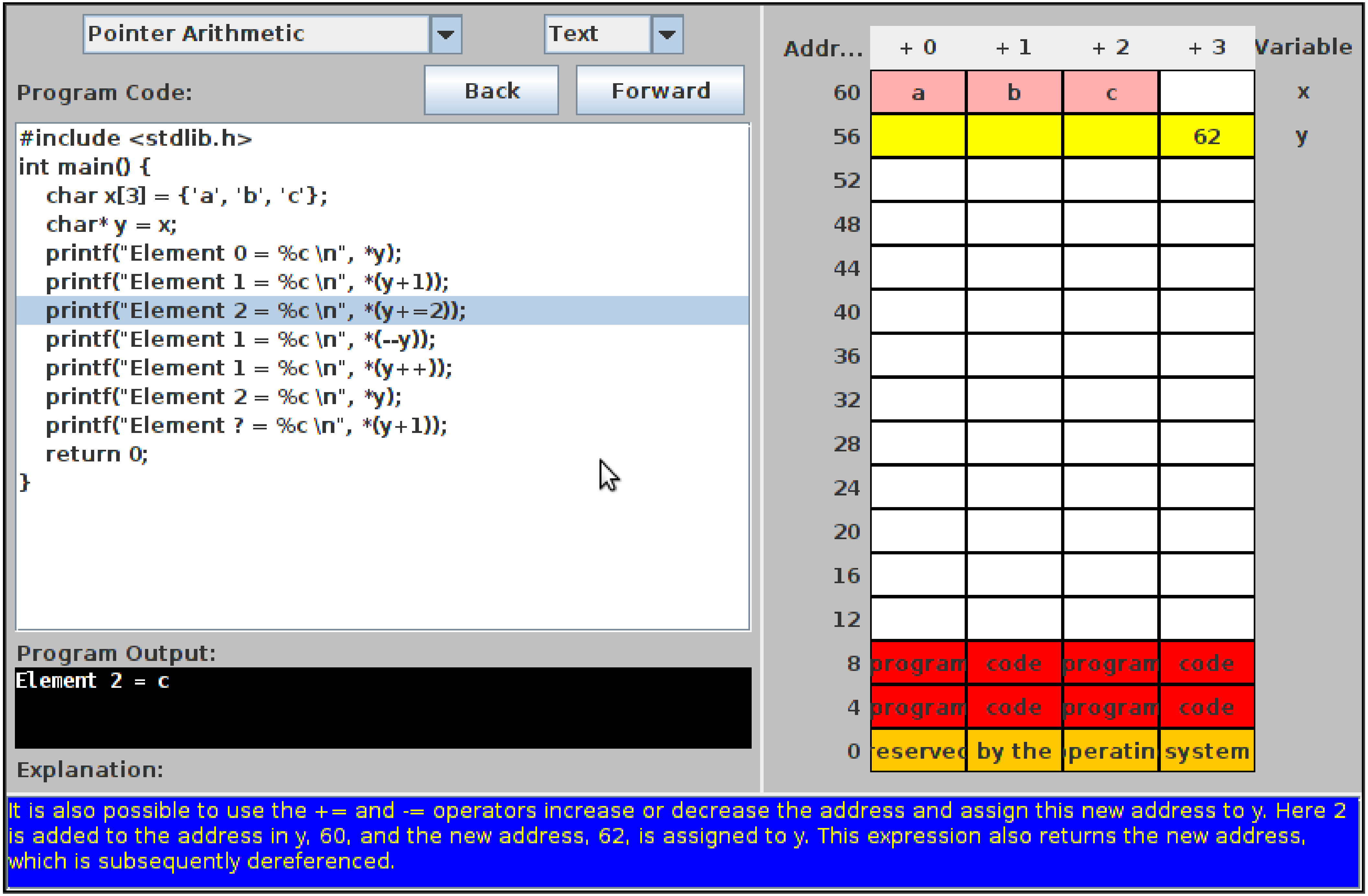
```
Element 1 = b
```

Explanation:

Here, the expression "y+1" returns the address 61 which is then dereferenced by * to return the value stored at 61.

Addr...	+ 0	+ 1	+ 2	+ 3	Variable
60	a	b	c		x
56				60	y
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	reserved	by the	operatin	system	

Check your answer; predict step 6



```
#include <stdlib.h>
int main() {
   char x[3] = {'a', 'b', 'c'};
   char* y = x;
   printf("Element 0 = %c \n", *y);
   printf("Element 1 = %c \n", *(y+1));
   printf("Element 2 = %c \n", *(y+=2));
   printf("Element 1 = %c \n", *(--y));
   printf("Element 1 = %c \n", *(y++));
   printf("Element 2 = %c \n", *y);
   printf("Element ? = %c \n", *(y+1));
   return 0;
}
```

Program Output:

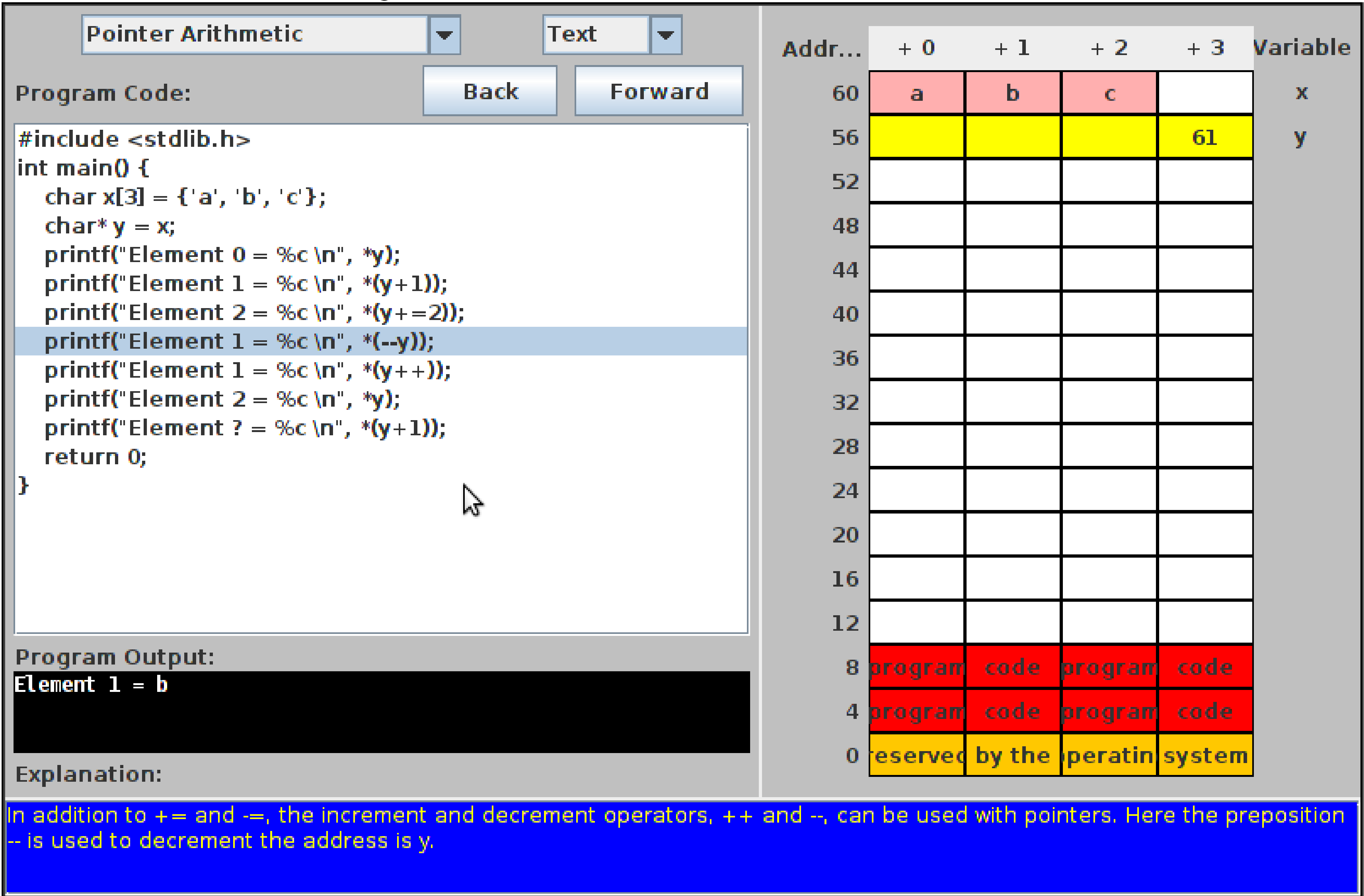
```
Element 2 = c
```

Addr...	+ 0	+ 1	+ 2	+ 3	Variable
60	a	b	c		x
56				62	y
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	eserved	by the	peratin	system	

Explanation:

It is also possible to use the += and -= operators increase or decrease the address and assign this new address to y. Here 2 is added to the address in y, 60, and the new address, 62, is assigned to y. This expression also returns the new address, which is subsequently dereferenced.

Check your answer; predict step 7



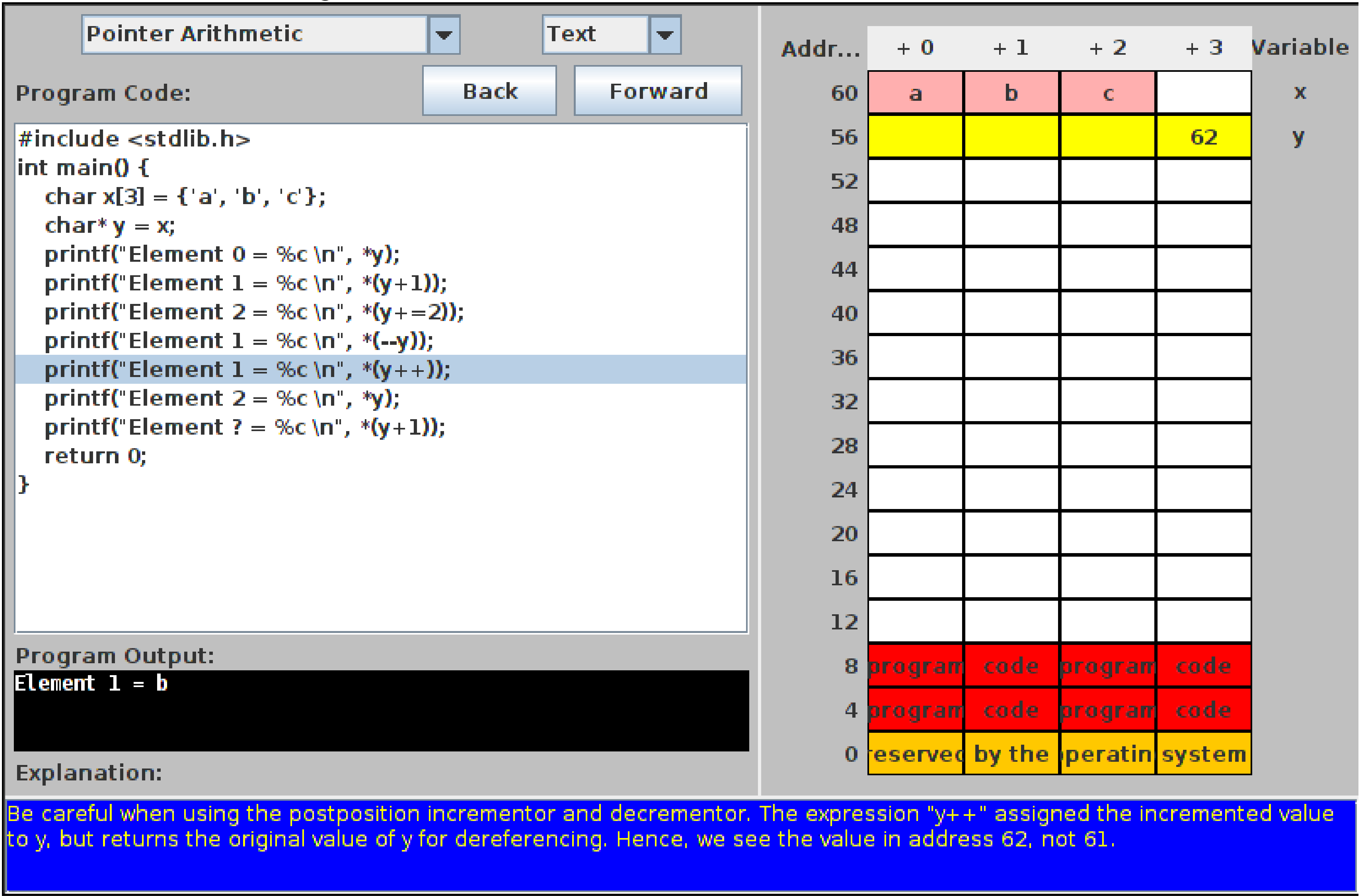
```
#include <stdlib.h>
int main() {
   char x[3] = {'a', 'b', 'c'};
   char* y = x;
   printf("Element 0 = %c \n", *y);
   printf("Element 1 = %c \n", *(y+1));
   printf("Element 2 = %c \n", *(y+=2));
   printf("Element 1 = %c \n", *(--y));
   printf("Element 1 = %c \n", *(y++));
   printf("Element 2 = %c \n", *y);
   printf("Element ? = %c \n", *(y+1));
   return 0;
}
```

```
Element 1 = b
```

Addr...	+ 0	+ 1	+ 2	+ 3	Variable
60	a	b	c		x
56				61	y
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	eserved	by the	peratin	system	

In addition to += and -=, the increment and decrement operators, ++ and --, can be used with pointers. Here the preposition -- is used to decrement the address is y.

Check your answer; predict steps 8&9



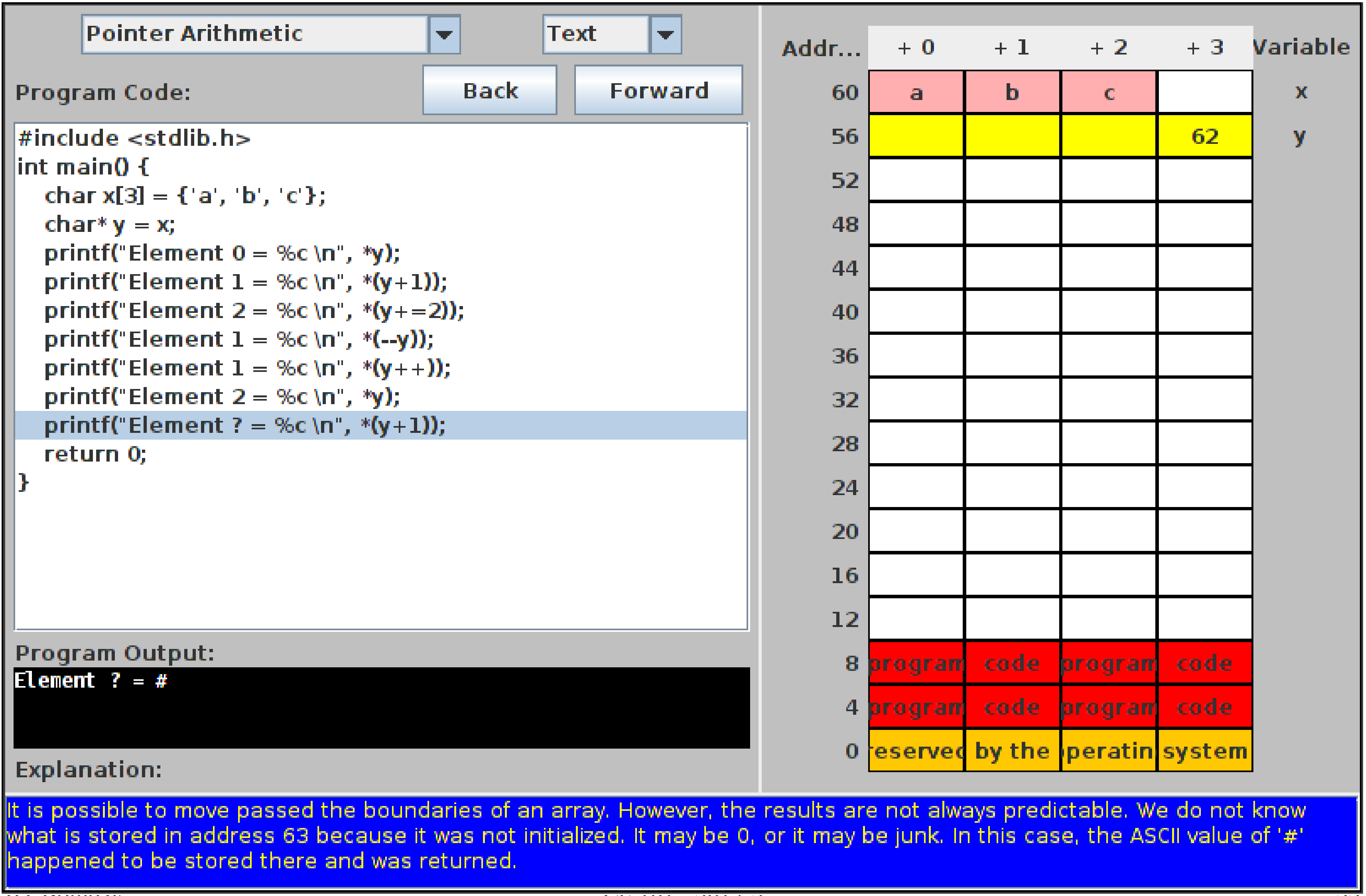
```
#include <stdlib.h>
int main() {
   char x[3] = {'a', 'b', 'c'};
   char* y = x;
   printf("Element 0 = %c \n", *y);
   printf("Element 1 = %c \n", *(y+1));
   printf("Element 2 = %c \n", *(y+=2));
   printf("Element 1 = %c \n", *(--y));
   printf("Element 1 = %c \n", *(y++));
   printf("Element 2 = %c \n", *y);
   printf("Element ? = %c \n", *(y+1));
   return 0;
}
```

```
Element 1 = b
```

Addr...	+ 0	+ 1	+ 2	+ 3	Variable
60	a	b	c		x
56				62	y
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	reserved	by the	operatin	system	

Be careful when using the postposition incrementor and decrementor. The expression "y++" assigned the incremented value to y, but returns the original value of y for dereferencing. Hence, we see the value in address 62, not 61.

How many of your predictions were right?



Program Code:

```
#include <stdlib.h>
int main() {
   char x[3] = {'a', 'b', 'c'};
   char* y = x;
   printf("Element 0 = %c \n", *y);
   printf("Element 1 = %c \n", *(y+1));
   printf("Element 2 = %c \n", *(y+=2));
   printf("Element 1 = %c \n", *(--y));
   printf("Element 1 = %c \n", *(y++));
   printf("Element 2 = %c \n", *y);
   printf("Element ? = %c \n", *(y+1));
   return 0;
}
```

Program Output:

```
Element ? = #
```

Explanation:

It is possible to move passed the boundaries of an array. However, the results are not always predictable. We do not know what is stored in address 63 because it was not initialized. It may be 0, or it may be junk. In this case, the ASCII value of '#' happened to be stored there and was returned.

Addr...	+ 0	+ 1	+ 2	+ 3	Variable
60	a	b	c		x
56				62	y
52					
48					
44					
40					
36					
32					
28					
24					
20					
16					
12					
8	program	code	program	code	
4	program	code	program	code	
0	eserved	by the	peratin	system	

In-class Tutorial Questions -1

Predict the output of the following program. Show the memory at the step indicated in comments.

```
int main () {
int firstvalue = 5, secondvalue = 15;
int *p1, *p2;
p1 = &firstvalue;  p2 = &secondvalue;
*p1 = 10;
*p2 = *p1; // Show the memory after this step is executed.
p1 = p2;
*p1 = 20;
cout << firstvalue << secondvalue << endl;
  return 0;
}
```

In-class Tutorial Question - 2

Your friend wrote the program below to print 573 and -123, and then print 573 and -123 again. However the output is not as expected. Identify the bug and fix it.

```
int main() {
  int M, N, A[3]; M = 573; N = -123;
  int *ptr1, *ptr2;
cout << M << " and " << N << endl;
ptr1 = &M; ptr2 = ptr1 + 1;
cout << *ptr1 << " " << *ptr2 << endl;
return 0;
}
```

In-class Tutorial Question - 3

Predict the output of the following program:

```
int main() {
int A[4], *p;

for (int i = 0; i < 4; i++) A[i] = i;
p = &A[0];
cout << *p << " " << *(p +=2) << *(p+1) + *(p-1) << endl;
return 0
}
```

More on pointers

- Which is correct? int* ptr; or int *ptr;
  - Spaces dont matter with operators and * is a unary operator, associates to right. So int* ptr should be read as int (*ptr);
  - *ptr is of type int. So ptr is of type address of int or int*

- int* iptr, jptr;

  declares iptr to be int* but jptr is int;

Note: Addresses of variable of type T can only be stored in pointers declared as type T*, even though all pointers are of 32 bits.

Reading Notes

- http://www.cplusplus.com/doc/tutorial/pointers/
- http://cslibrary.stanford.edu/102/
  - Read Section 1
  - Also has the link to the Binky video
- Self-study: “pointer to pointer” visualization
  - http://www2.sis.pitt.edu/~is2470pb/Fall02/Final/rm/interactive.html