Week 7

Linked Lists

Notes about Projects

- Submit your .java files in a .zip (Not a .jar)
- Put your tests into the package edu.iastate.cs228.hwx.test
- PLEASE PLEASE compile and run your code before submitting

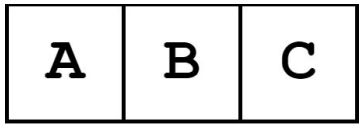
Linked Lists

What is a Linked List?

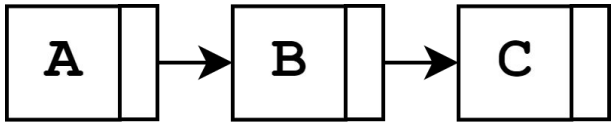
Ordered data structure which stores data in **Nodes**

Nodes point to the next ordered element in the list

Array

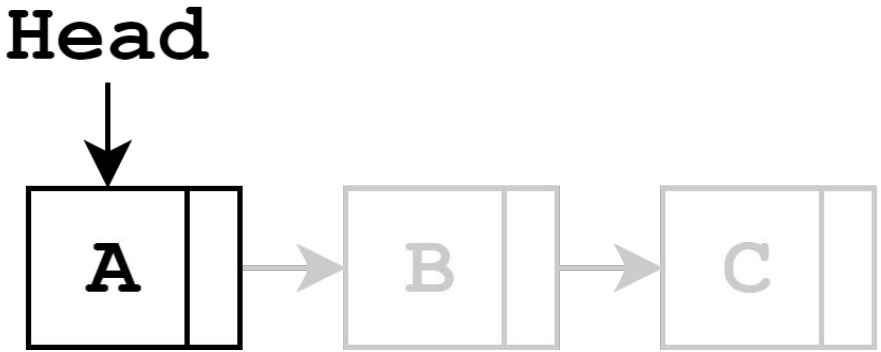


Linked List



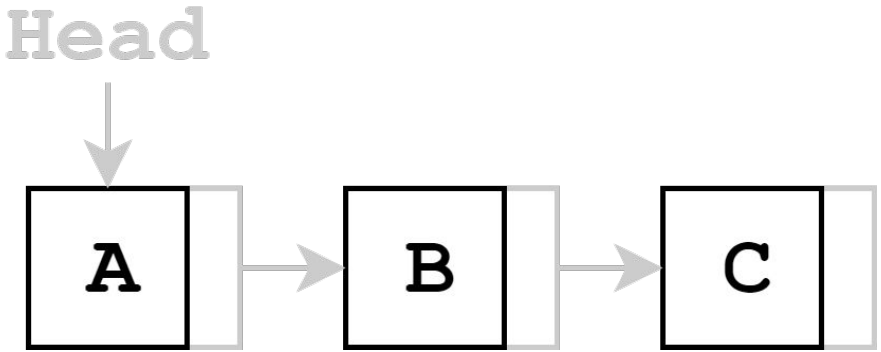
What is a Linked List?

The Linked List starts at the **head**



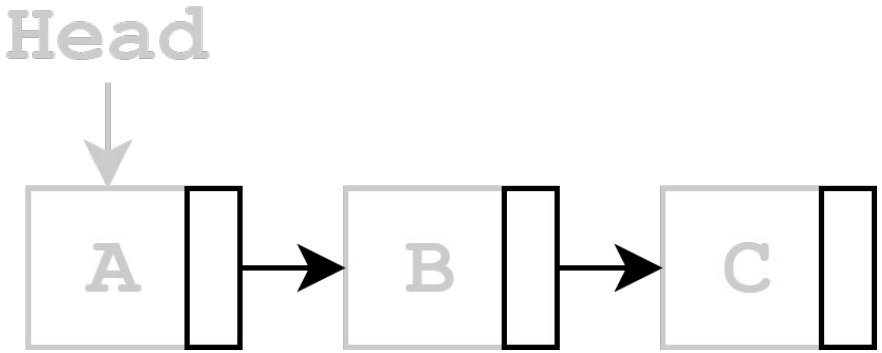
What is a Linked List?

Each node of the linked list stores **data**



What is a Linked List?

Each node of the Linked List will point to the **next** node



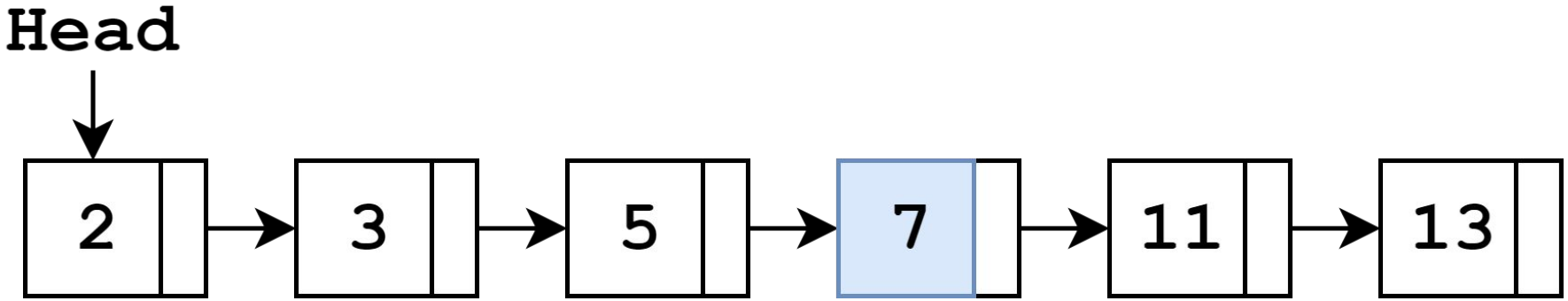
Linked List Class

```
class LinkedList<T>
{
    Node head;

    class Node
    {
        Node next;
        T data;
    }
}
```

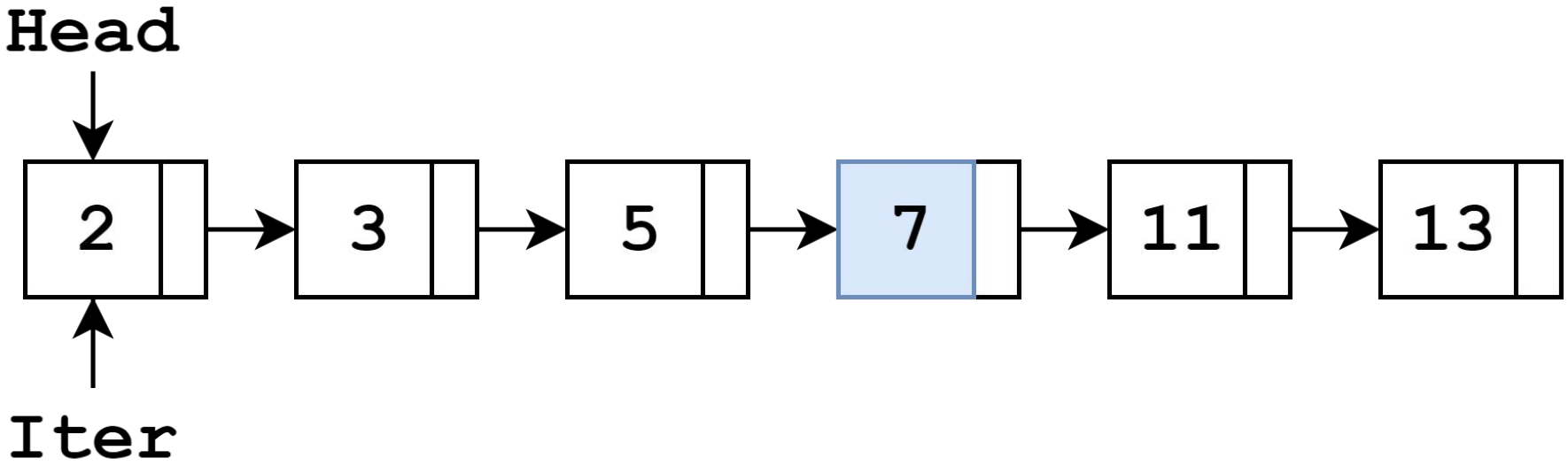
Linked List Operations: Get

```
char c = myLinkedList.get(3);
```



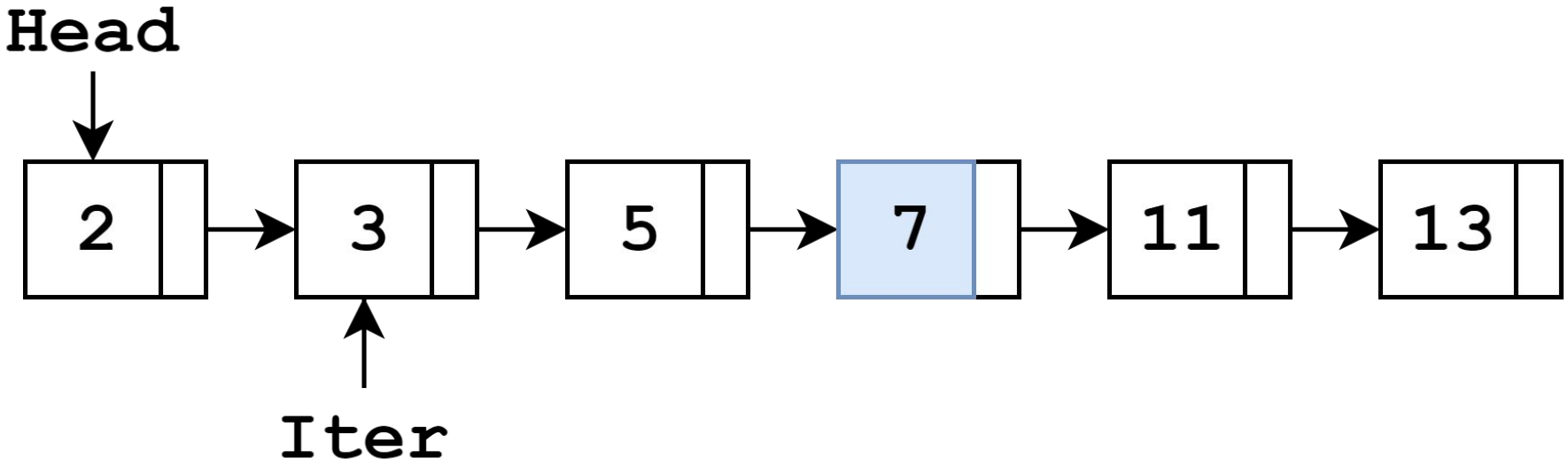
Linked List Operations: Get

```
Node iter = head;
```



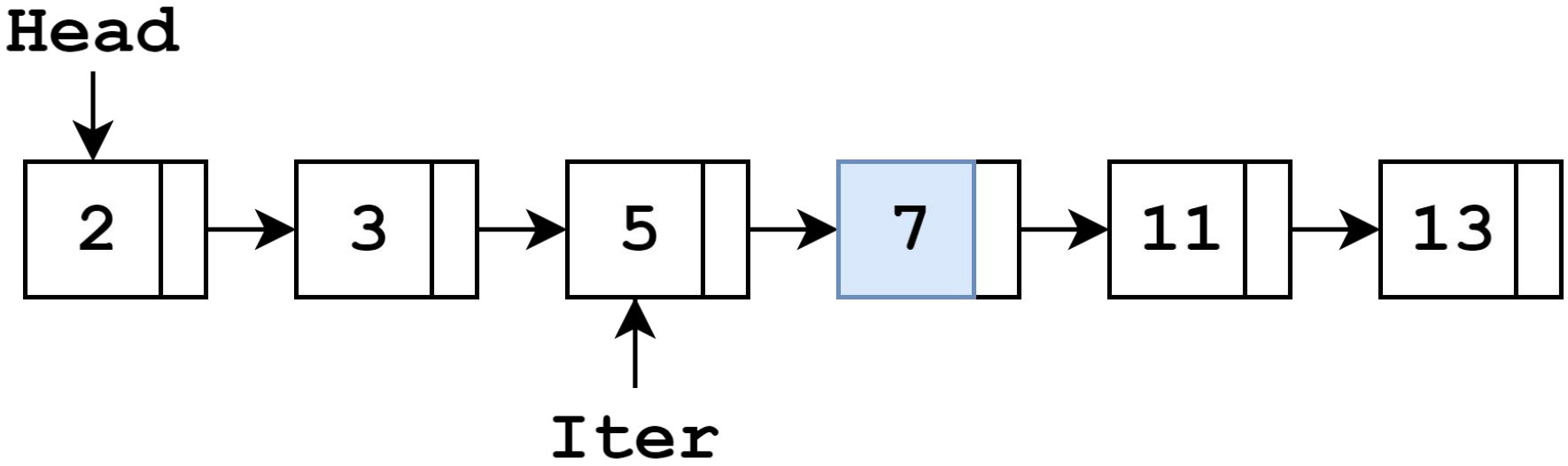
Linked List Operations: Get

```
iter = iter.next;
```



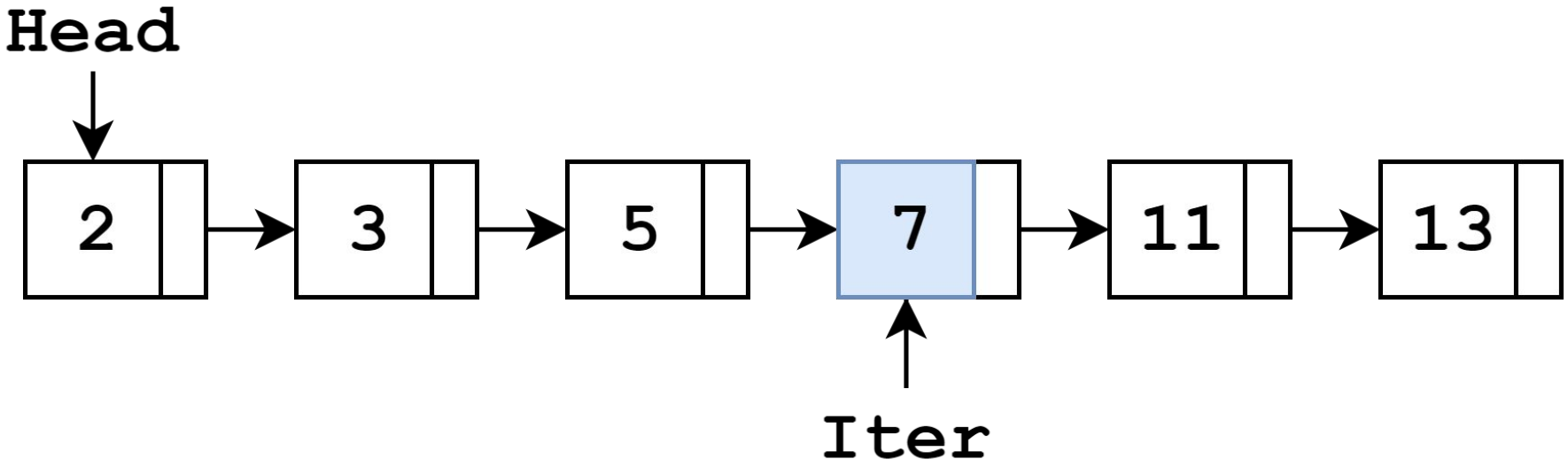
Linked List Operations: Get

```
iter = iter.next;
```



Linked List Operations: Get

```
iter = iter.next;
```

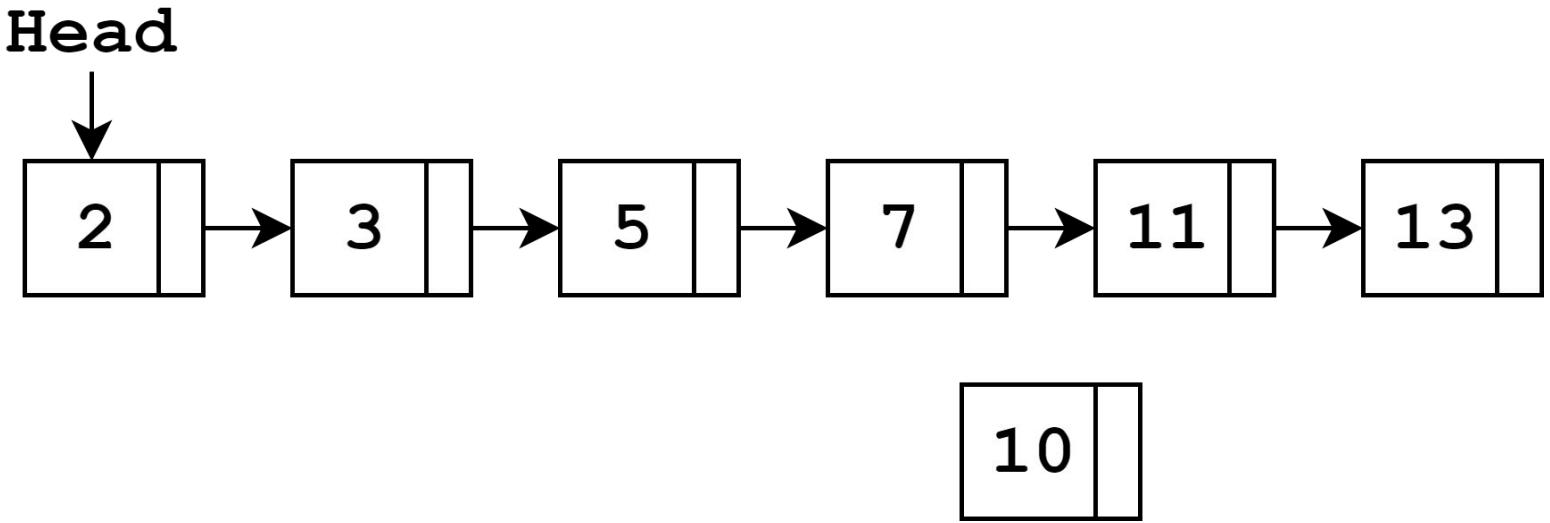


Linked List Operations: Get

```
public T get(int index)
    iter = head
    for index # of times
        iter = iter.next
    return iter.data
```

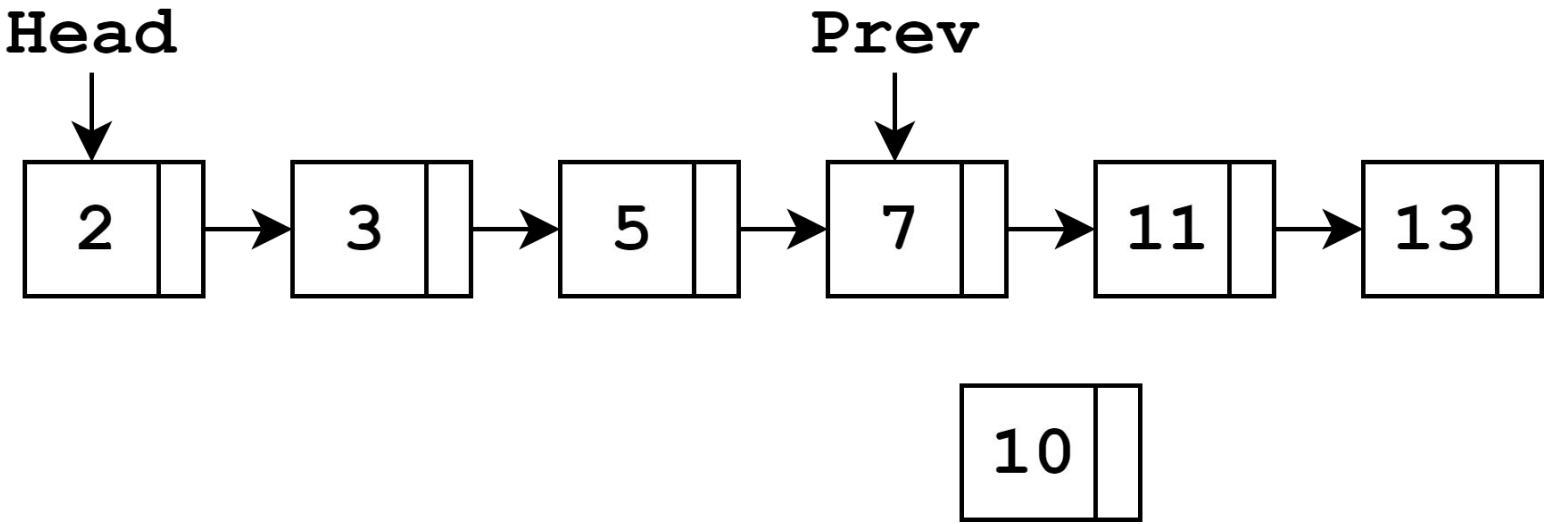
Linked List Operations: Insert

```
myLinkedList.insert(10, 4);
```



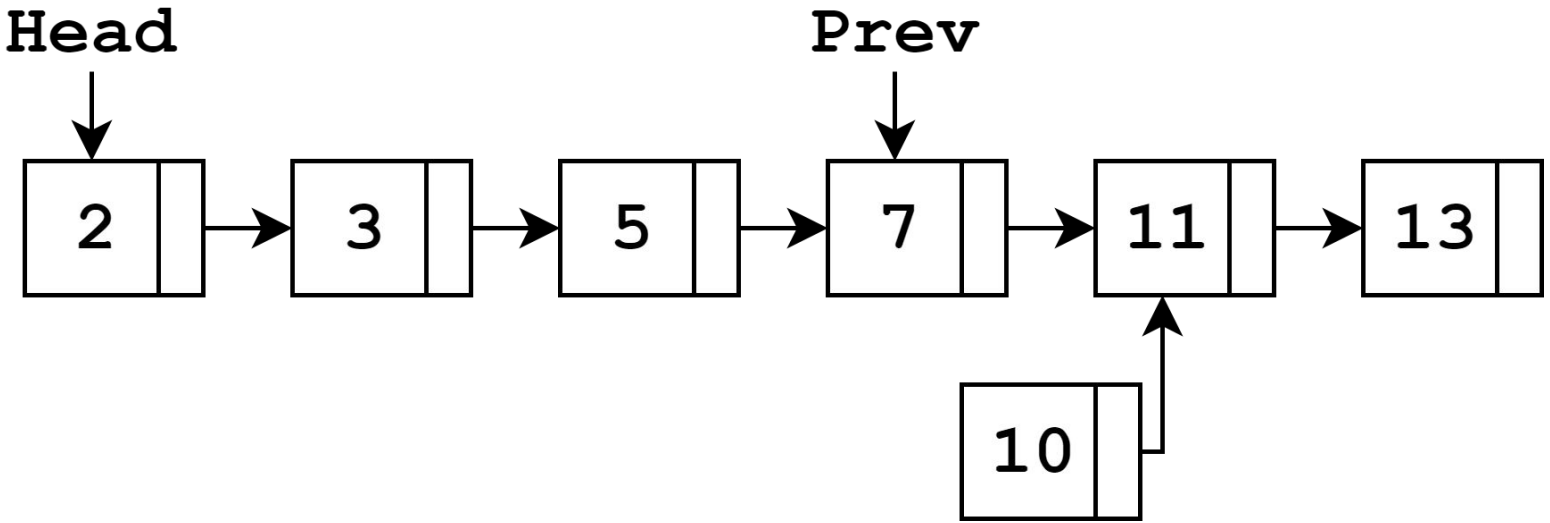
Linked List Operations: Insert

```
Node prev = getNodeAt(index - 1);
```



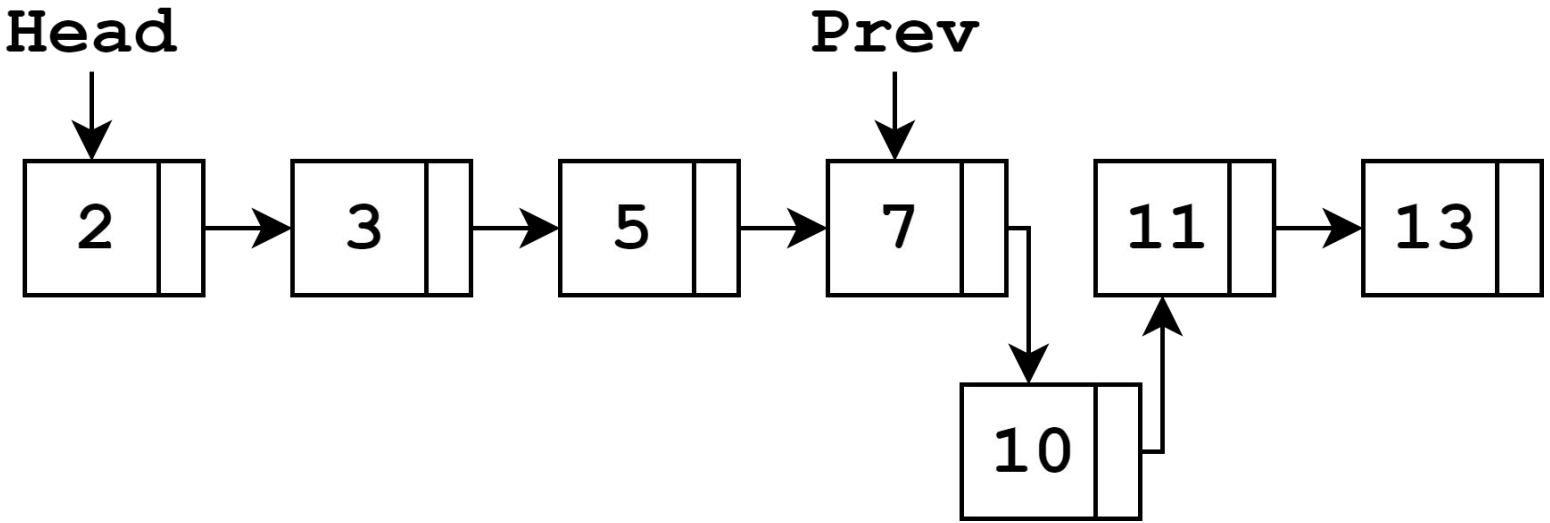
Linked List Operations: Insert

```
newNode.next = prev.next;
```



Linked List Operations: Insert

```
prev.next = newNode;
```

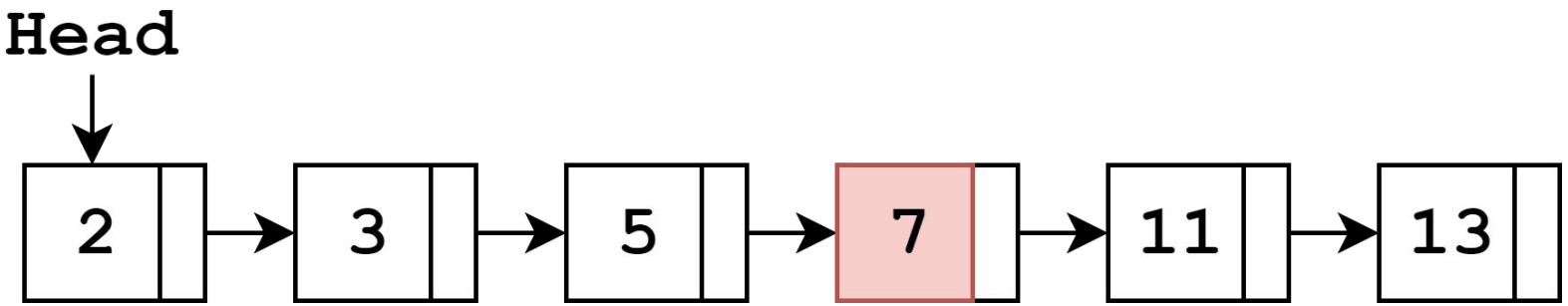


Linked List Operations: Insert

```
public void insert(T data, int index)
    n = new Node(data)
    if index is 0
        n.next = head
        head = n
    else
        prev = The node at index - 1
        n.next = prev.next
        prev.next = n
```

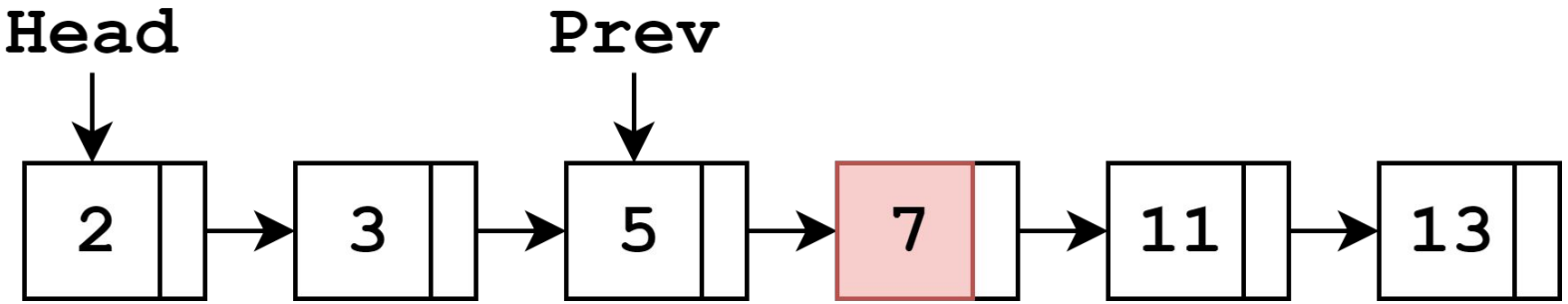
Linked List Operations: Remove

```
myLinkedList.remove(3);
```



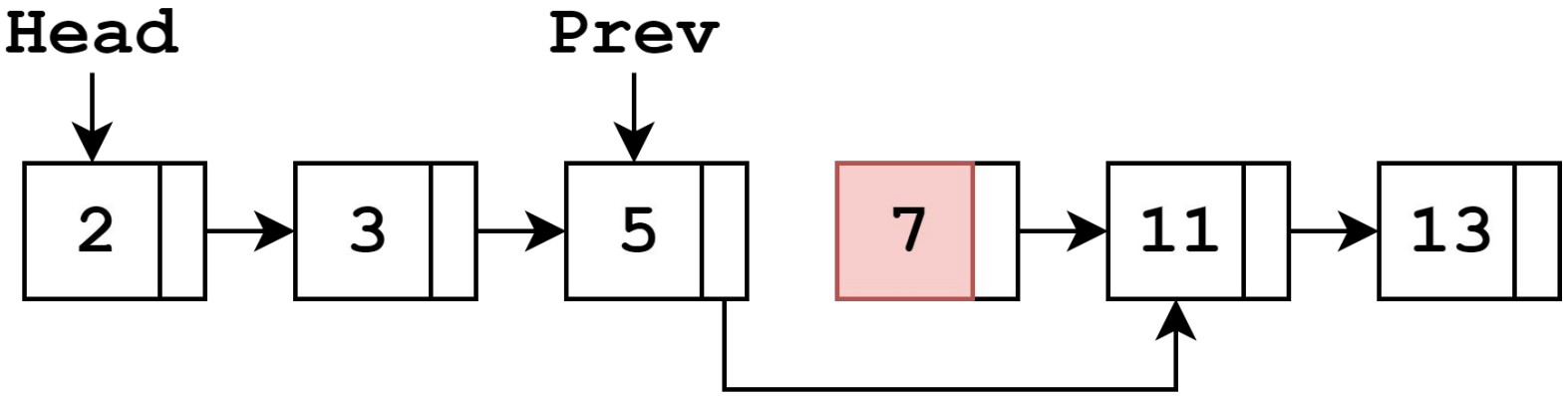
Linked List Operations: Remove

```
Node prev = getNodeAt(index - 1);
```



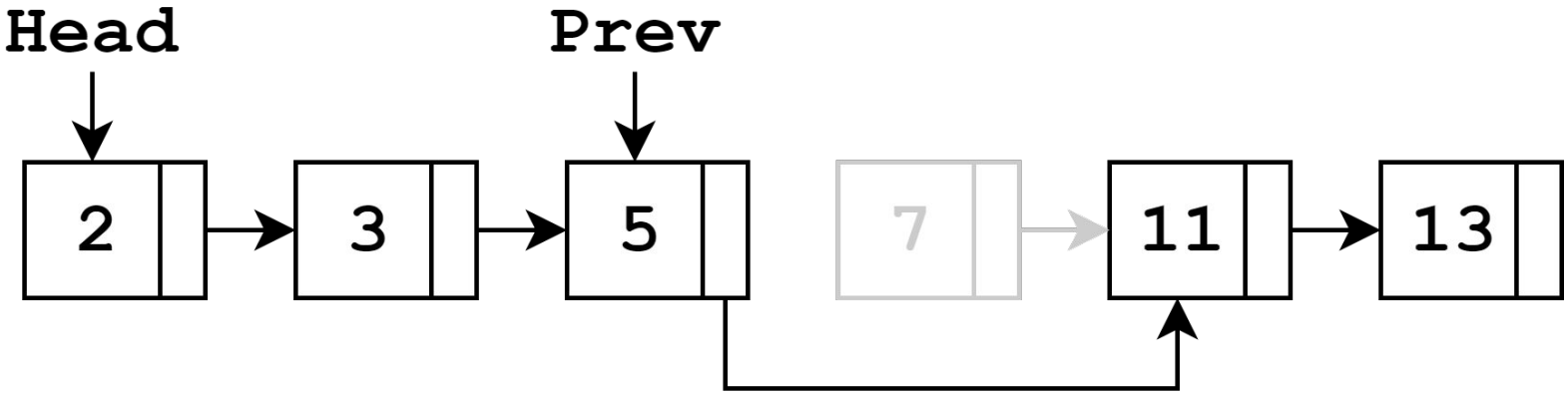
Linked List Operations: Remove

```
prev.next = prev.next.next;
```



Linked List Operations: Remove

Since 7 is no longer referenced, Java automatically deletes it.



Linked List Operations: Remove

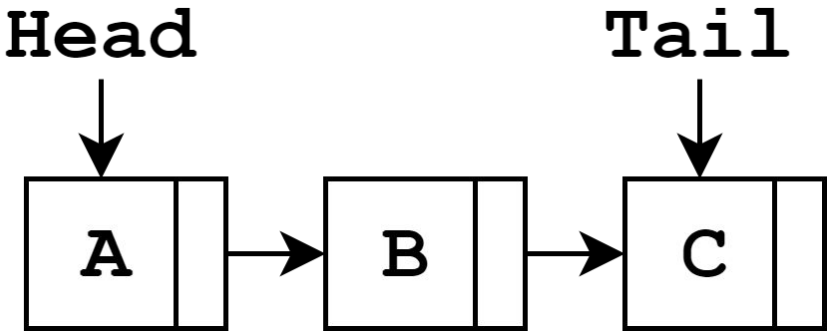
```
public T remove(int index)
    if index is 0
        data = head.data
        head = head.next;
    else
        prev = The node at index - 1
        data = prev.next.data
        prev.next = prev.next.next
    return data
```

Linked List Variations

Linked List Tail

A reference in Linked List pointing to last element.

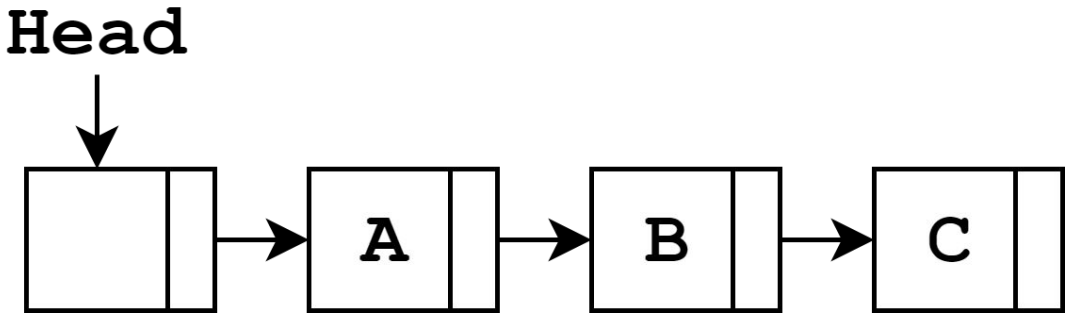
O(1) to add node to end of list, O(n) to remove from end of list.



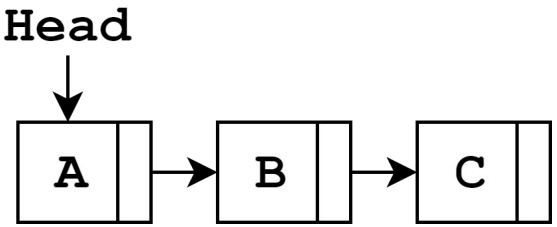
Dummy Nodes

Head always points to a node that never contains data.

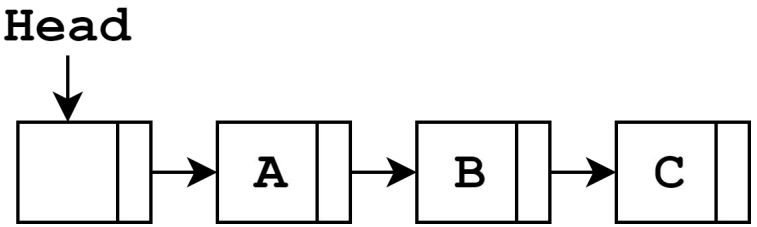
List starts after dummy node.



Dummy Nodes: But why?



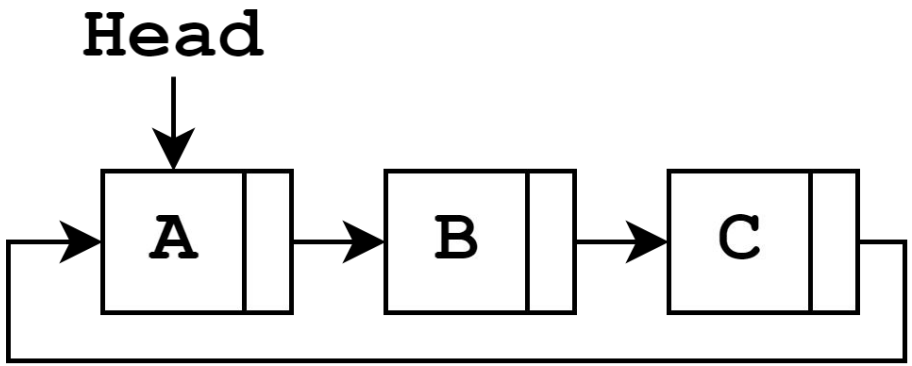

```
public void insert(T data, int i)
     n = new Node(data)
     if index is 0
          n.next = head
          head = n
     else
          prev = The node at i-1
          n.next = prev.next
          prev.next = n
```




```
public void insert(T data, int i)
     n = new Node(data)
     prev = The node at i-1
     n.next = prev.next
     prev.next = n
```

Circular Linked List

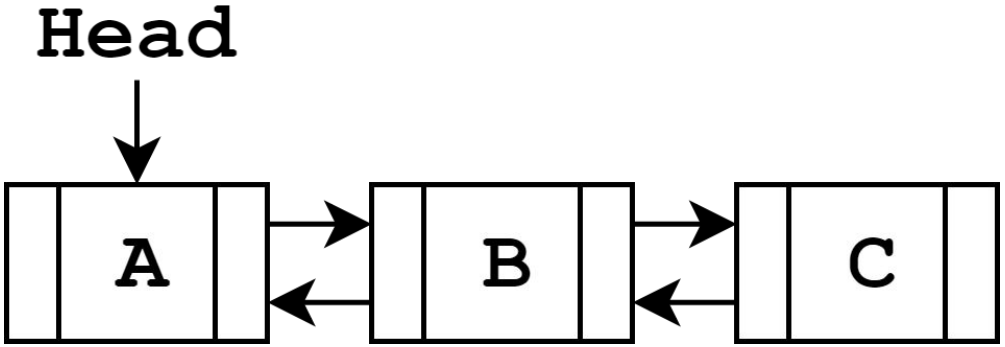
The last node in the list points to the first.



Doubly Linked List

Every node refers to the **previous** node as well as the next one.

Faster to search, easier to insert/remove, but takes extra memory.



Linked Lists vs. Arrays

Linked List	Array (or ArrayList)
get:	get:
search:	search:
insert:	insert:
remove:	remove:

Linked List		Array (or ArrayList)	
get:	**O(n)**	get:	**O(1)**
search:		search:	
insert:		insert:	
remove:		remove:	

	Linked List	Array (or ArrayList)
get:	**O(n)**	**O(1)**
search:	**O(n)**	**O(n)**
insert:		
remove:		

	Linked List		Array (or ArrayList)
get:	**O(n)**	get:	**O(1)**
search:	**O(n)**	search:	**O(n)**
insert:	**O(n)***	insert:	**O(n)**
remove:		remove:	

	Linked List		Array (or ArrayList)
get:	**O(n)**	get:	**O(1)**
search:	**O(n)**	search:	**O(n)**
insert:	**O(n)***	insert:	**O(n)**
remove:	**O(n)***	remove:	**O(n)**

	Linked List		Array (or ArrayList)
get:	**O(n)**	get:	**O(1)**
search:	**O(n)**	search:	**O(n)**
insert:	**O(n)***	insert:	**O(n)**
remove:	**O(n)***	remove:	**O(n)**

*O(1) if insert/removing at beginning or end

Linked List

PROS

- Quick insertion/deletion (most times)

CONS

- Up to twice as much memory
- Slow to get node at an index

Array (or ArrayList)

PROS

- Quick access to elements

CONS

- Slow insertion/deletion
- If extra space needed, all elements must be moved

Interview Questions

Buy the book for $40: https://www.amazon.com/Cracking-Coding-Interview-Programming-Questions/dp/0984782850

Problem 1

Write code to reverse a linked list in place. Return the new head node.

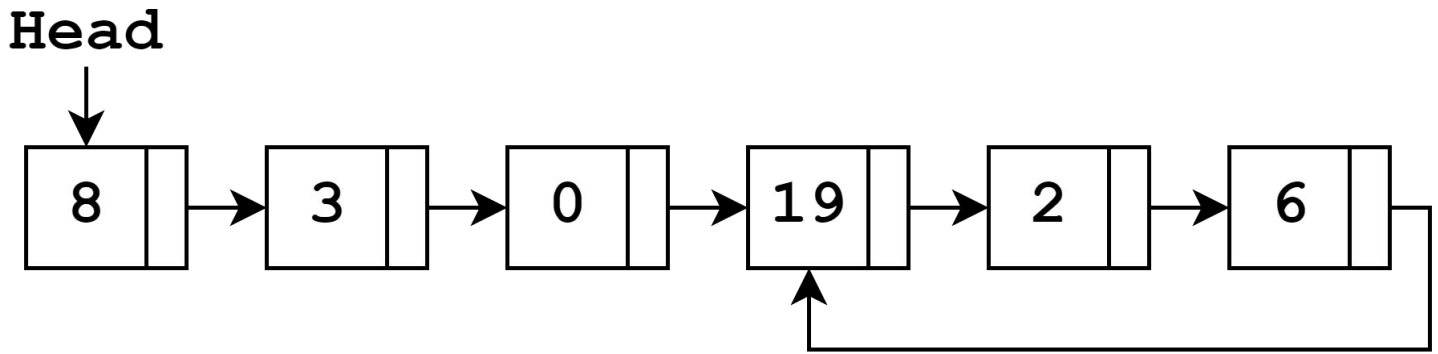
Don’t use any additional buffers (e.g., arrays).

Problem 1 Solution

```
Node reverseLinkedList(Node head)
    r = Empty linked list
    while head is not null
        n = First node removed from list at head
        Add n to beginning of list at r
    return r
```

Problem 2

How would you determine if a linked list has a cycle or not?



Problem 2 Solution

```
boolean hasCycle(Node head)
    fast = head
    slow = head
    while fast is not null
        fast = fast.next.next
        slow = slow.next
        if fast is the same node as slow
            return true
    return false
```

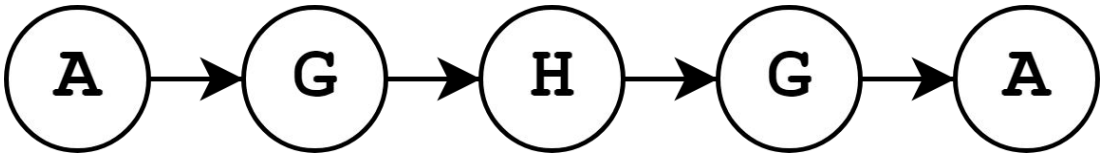
Problem 3

Write code to remove duplicates from a linked list

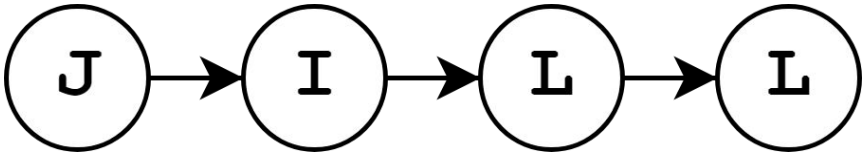
Problem 4

Implement a function to check if a linked list is a palindrome

Palindrome



Not Palindrome



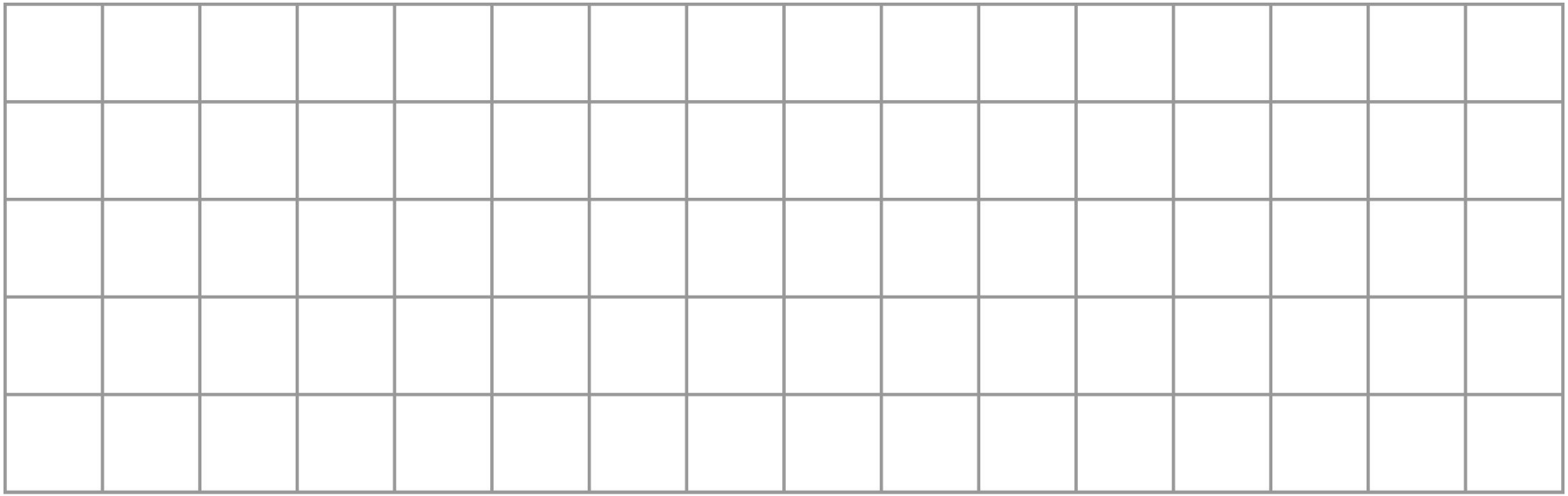
Found in Cracking the Coding Interview: page 95, problem 2.6

End

Bonus/Omitted Material

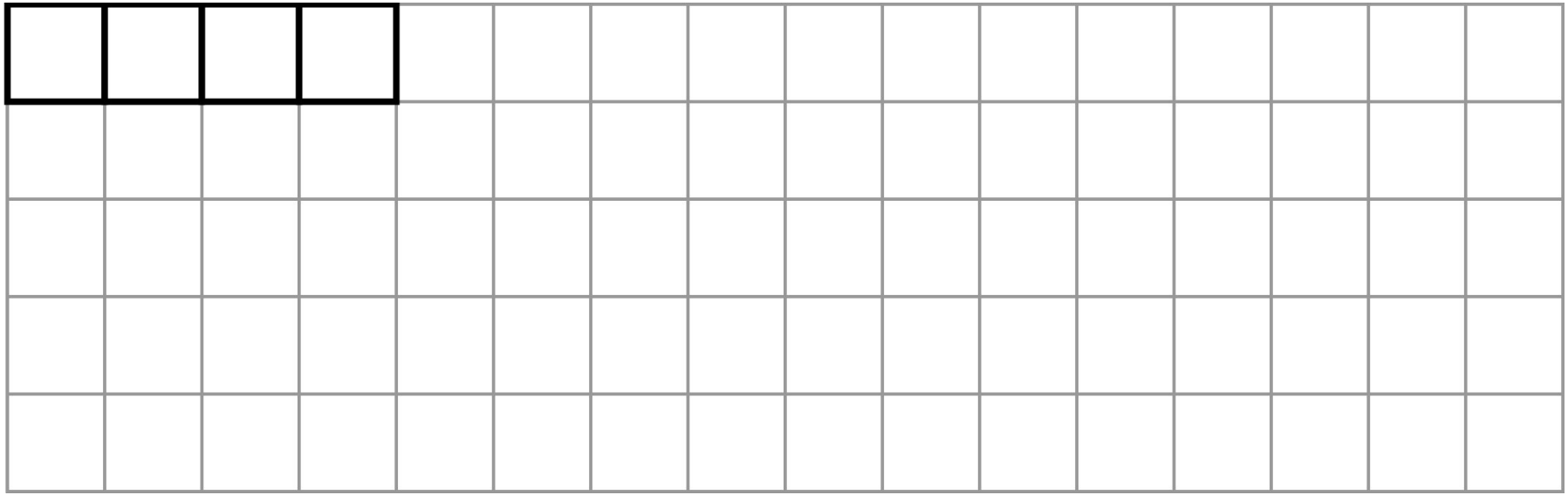
Issues with Arrays

What's the problem with Arrays?



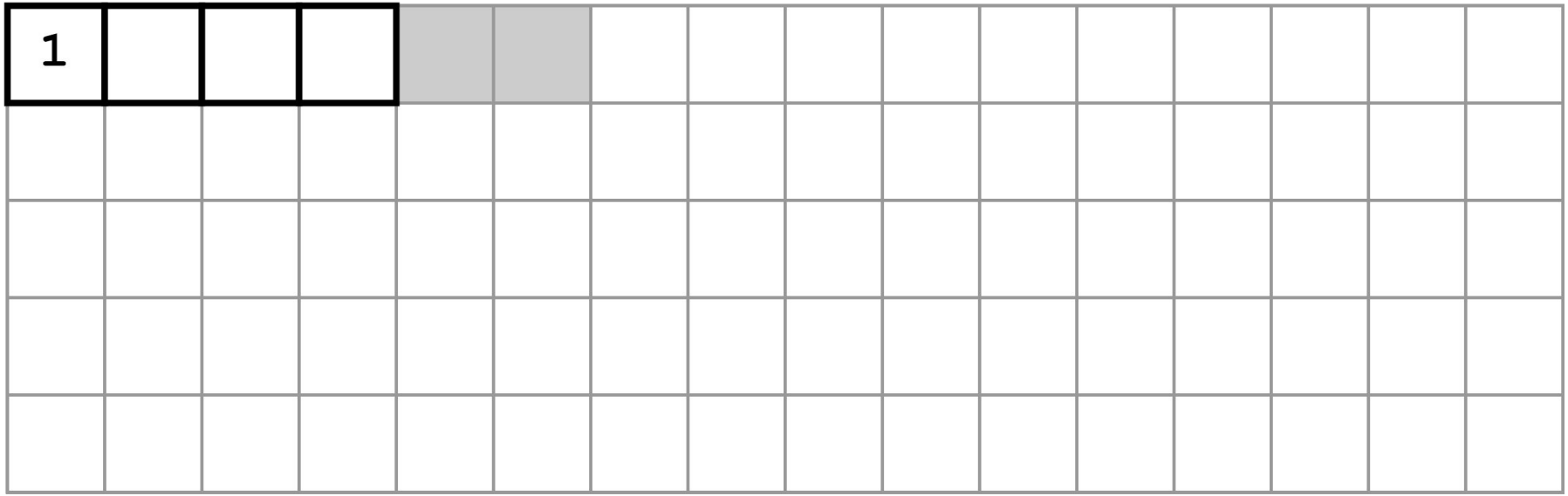
Program Memory

```
Integer[] arr = new Integer[4];
```



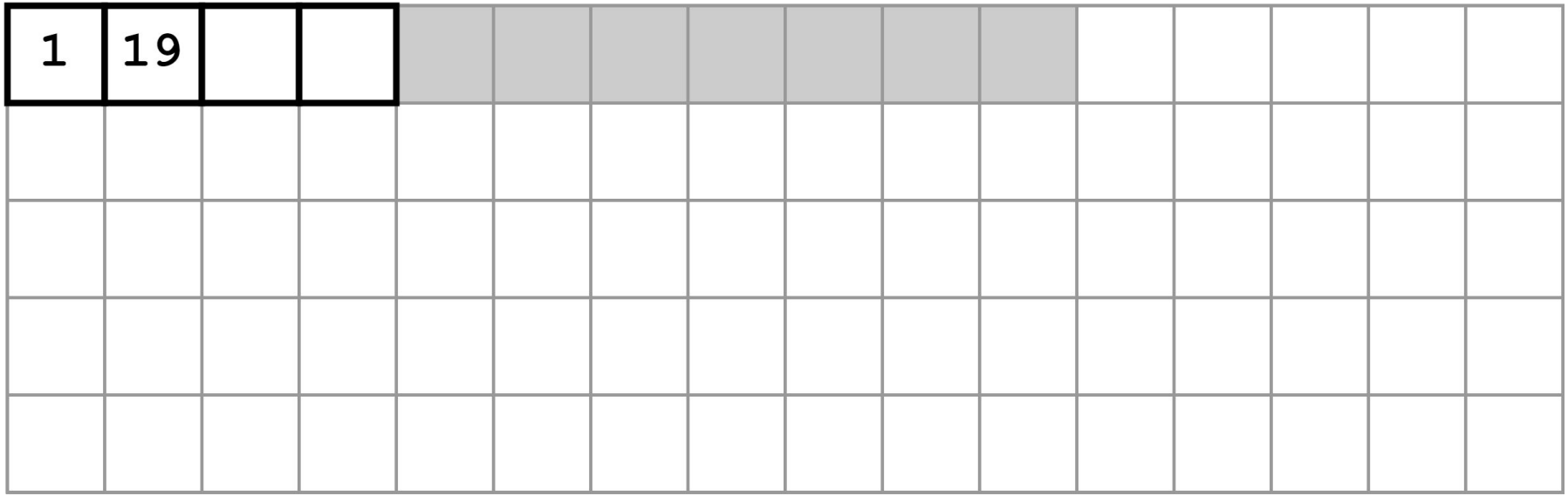
Program Memory

```
arr[0] = 1;
```



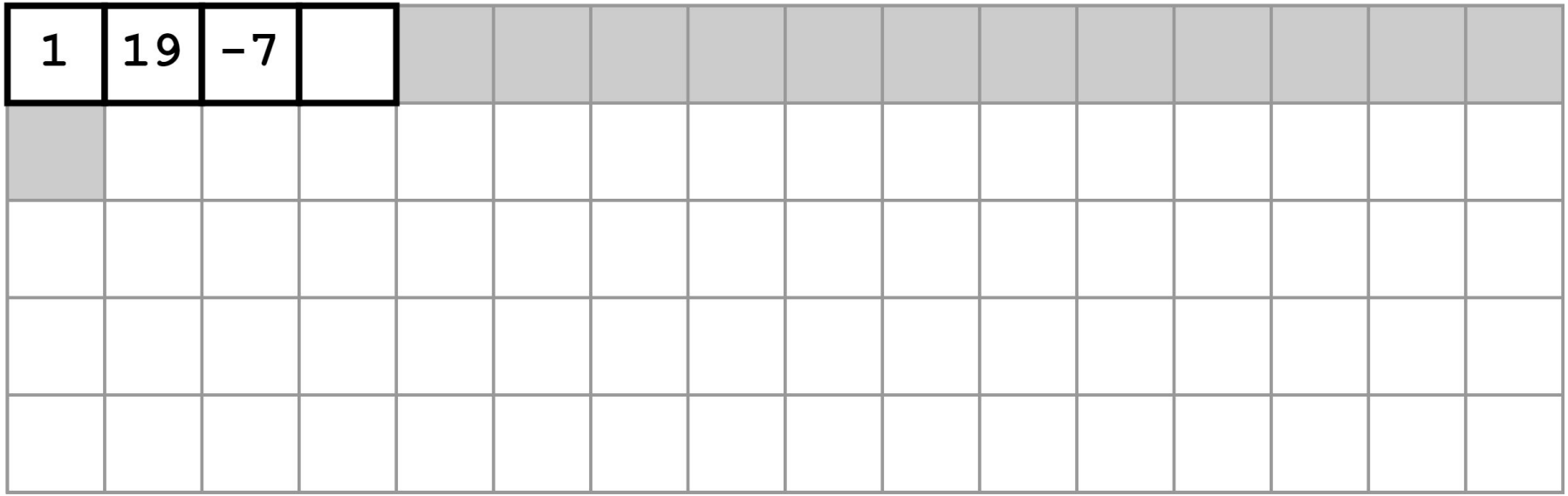
Program Memory

```
arr[1] = 19;
```



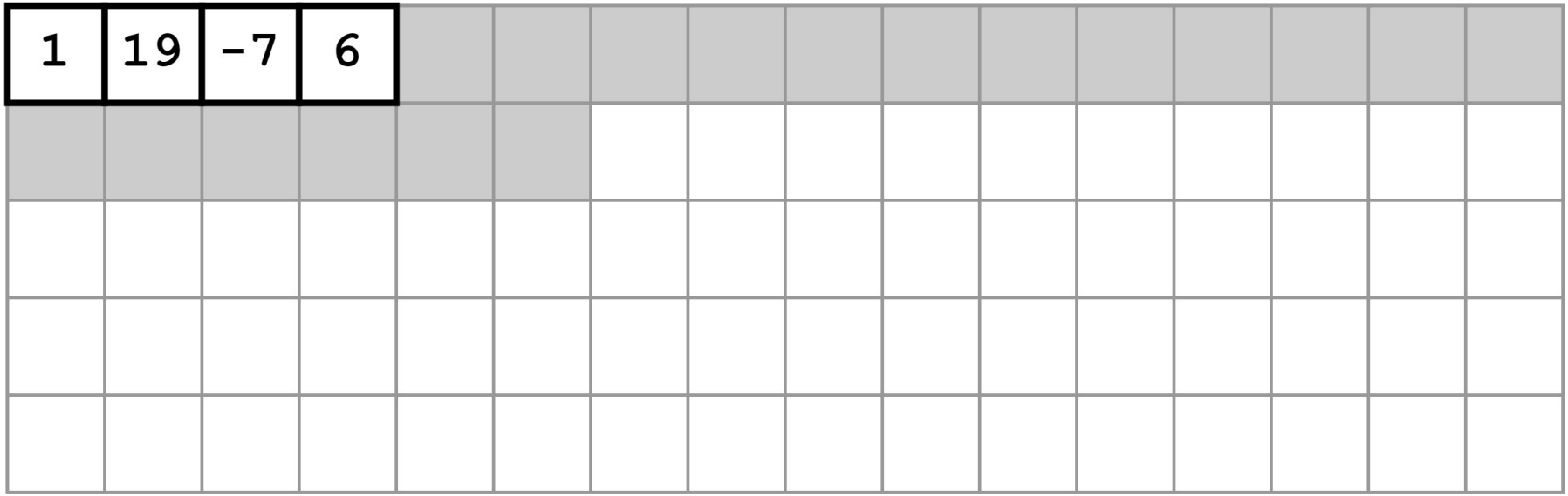
Program Memory

```
arr[1] = -7;
```



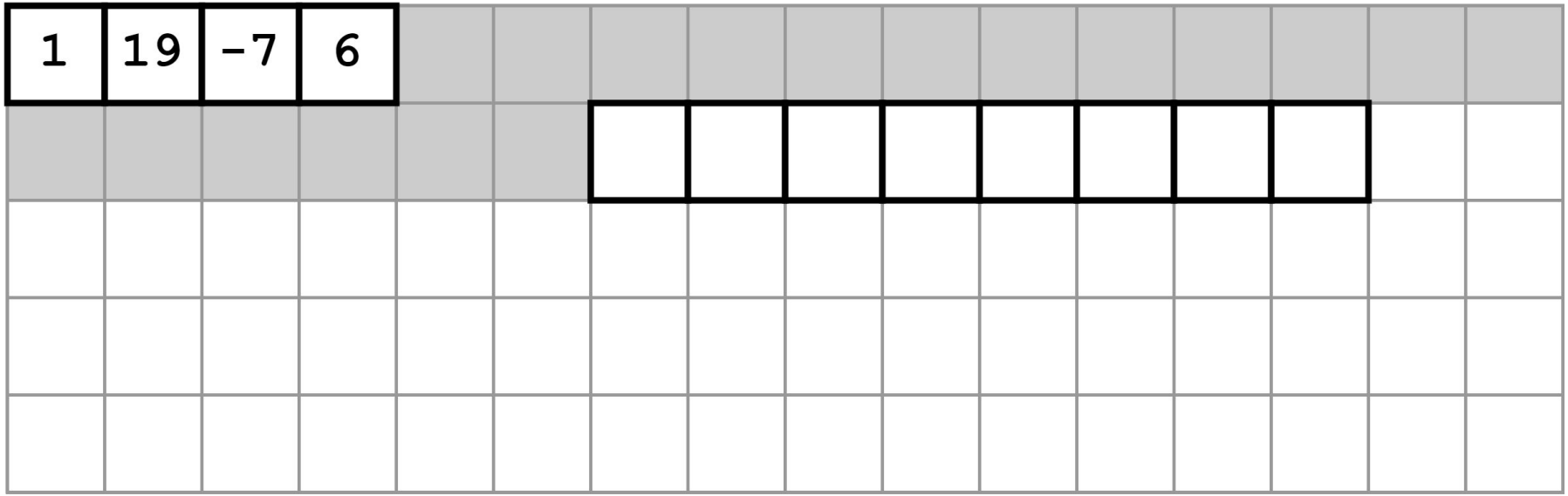
Program Memory

```
arr[1] = 6;
```



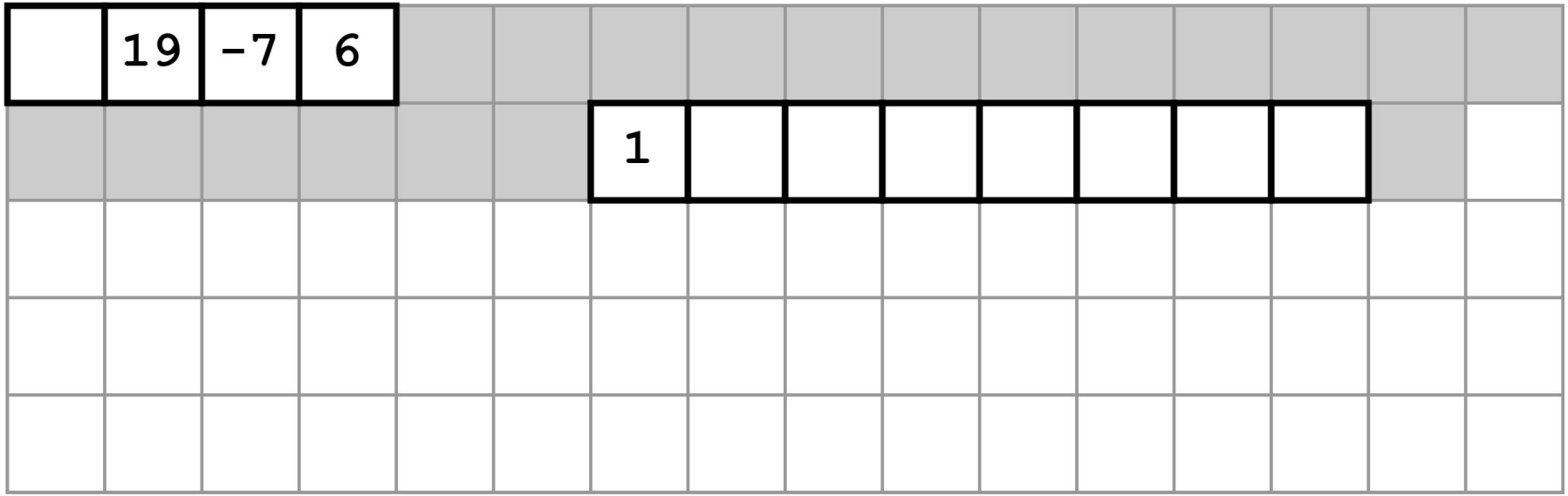
Program Memory

```
Integer[] temp = new Integer[8];
```

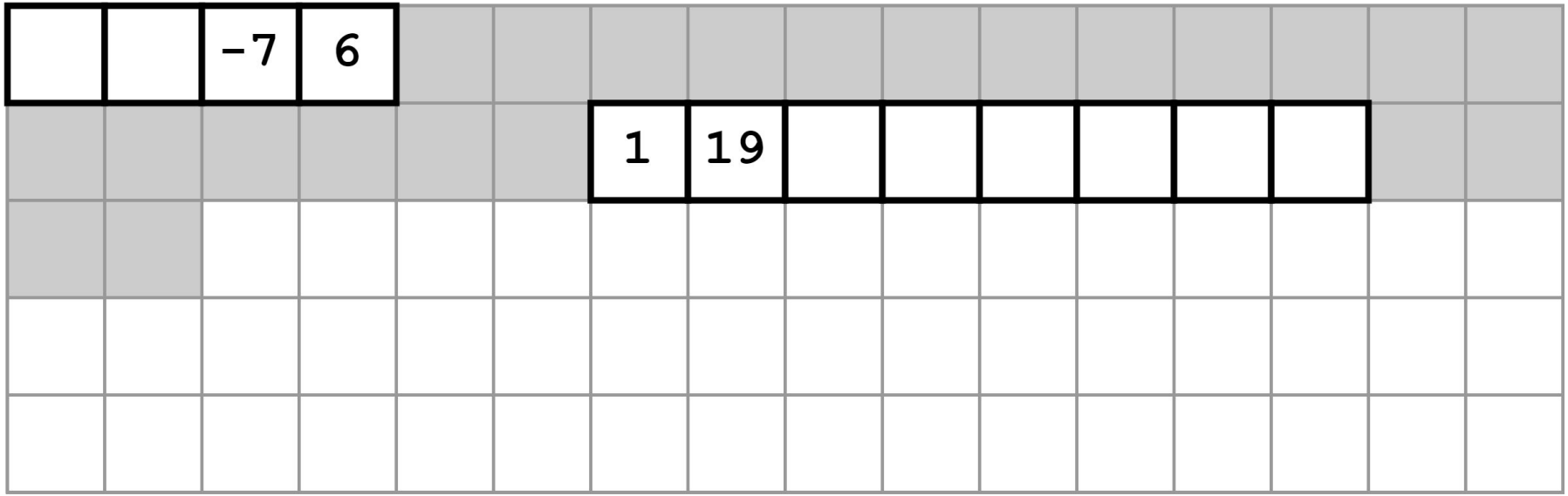


Program Memory

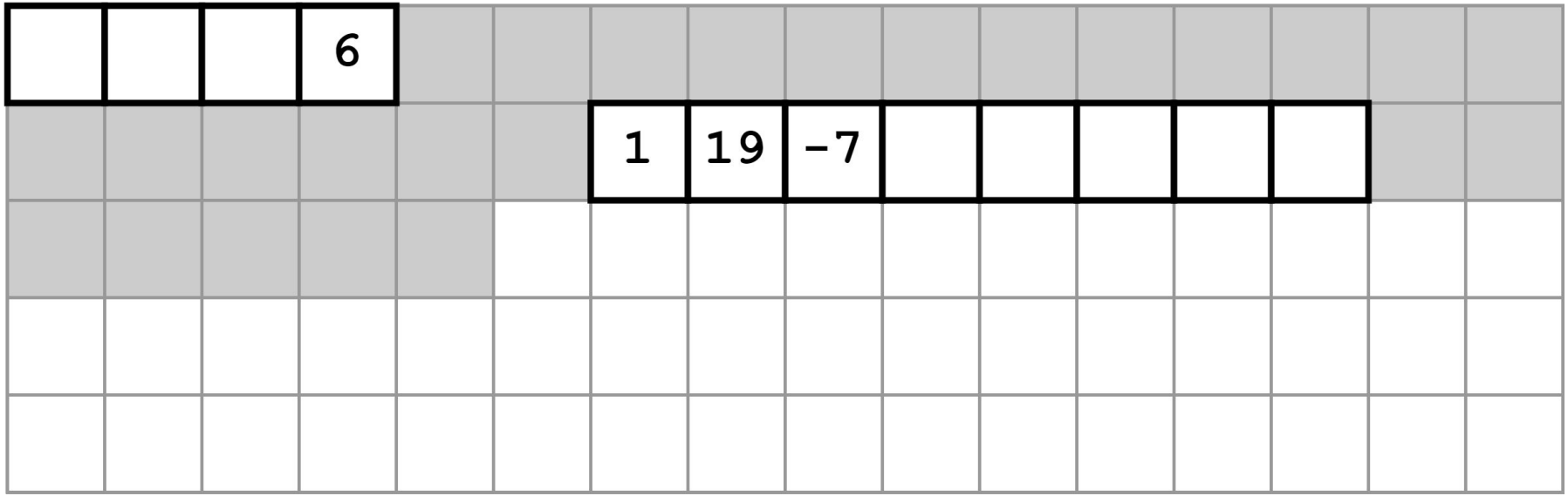
```
temp[0] = arr[0];
```



```
temp[1] = arr[1];
```

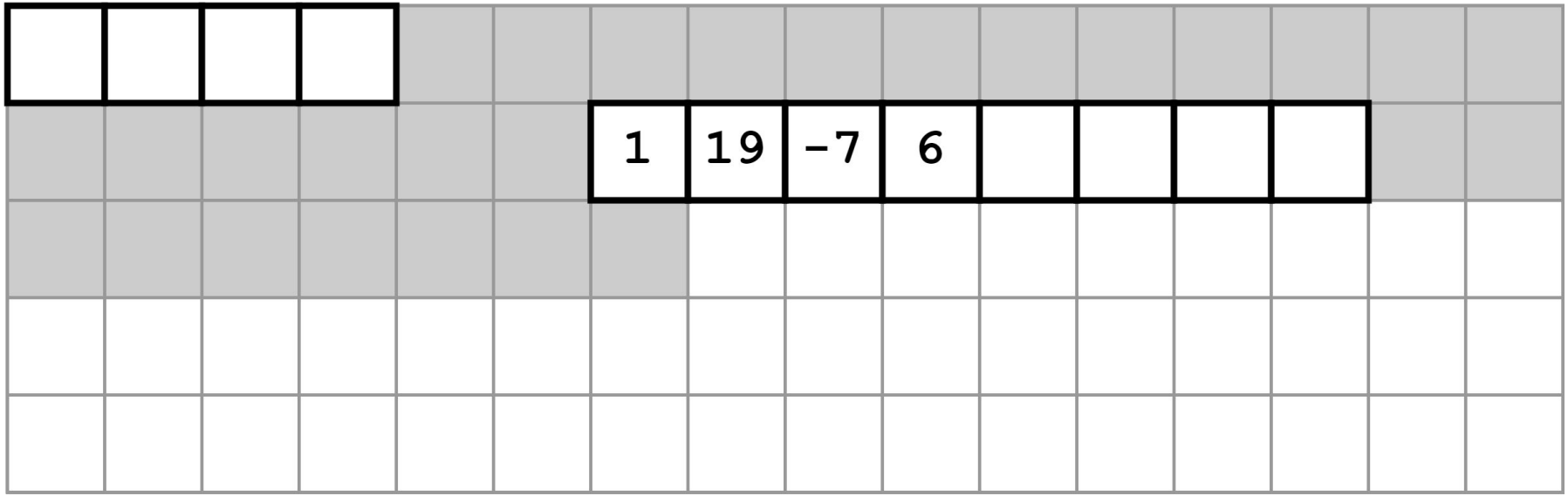


```
temp[2] = arr[2];
```



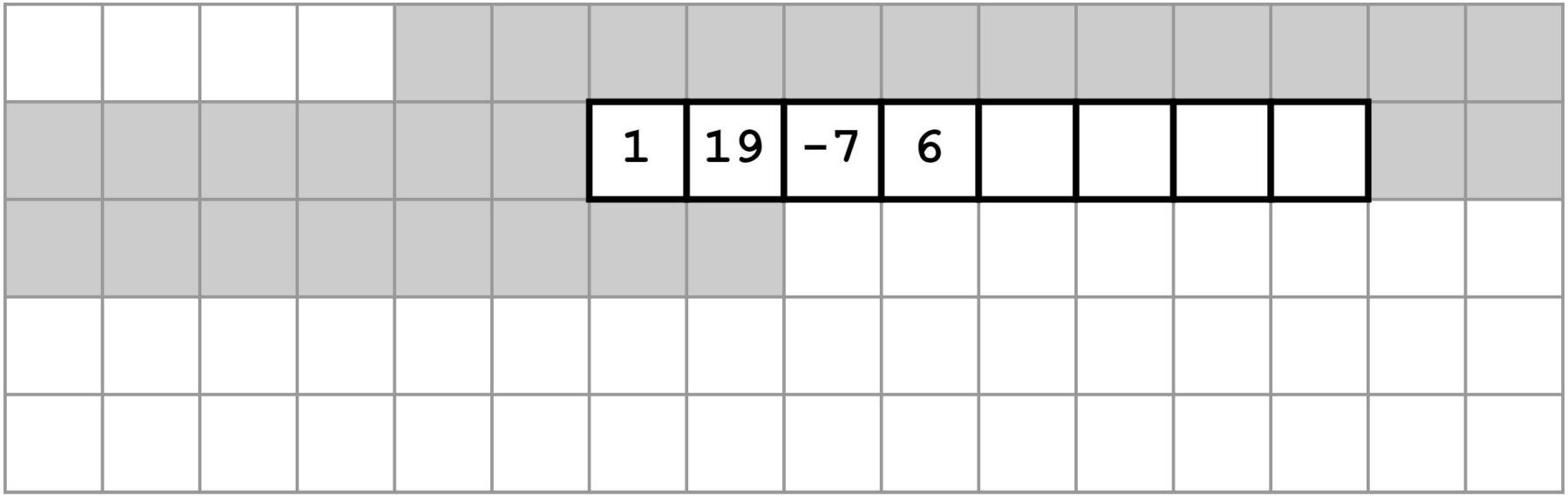
Program Memory

```
temp[3] = arr[3];
```



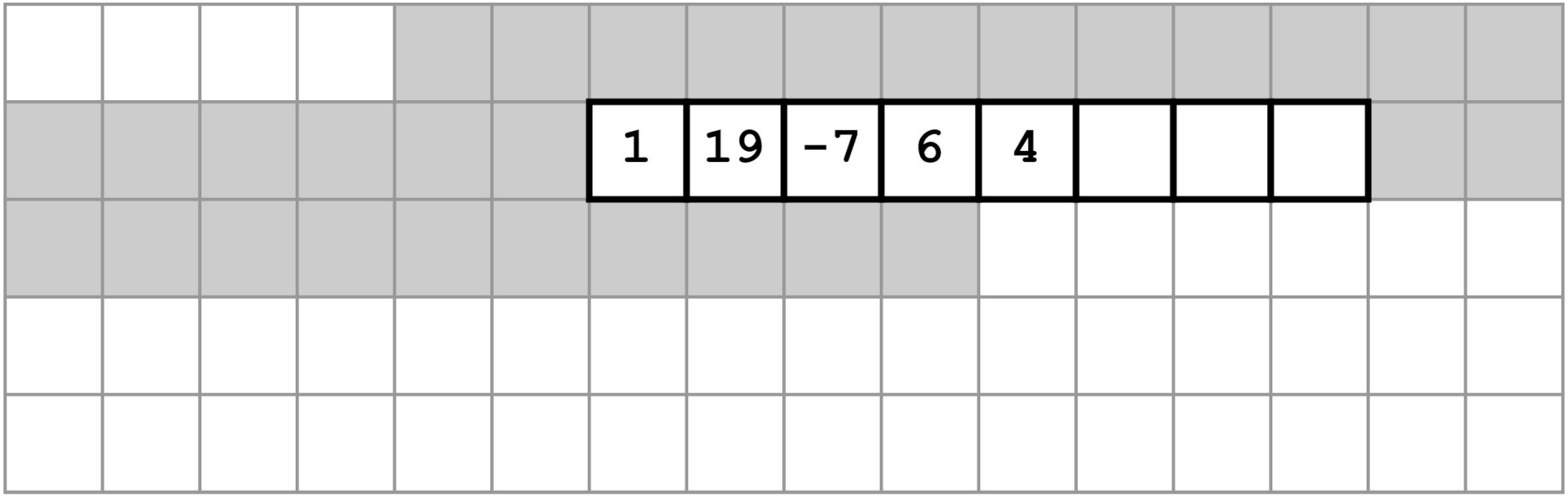
Program Memory

```
arr = temp;
```



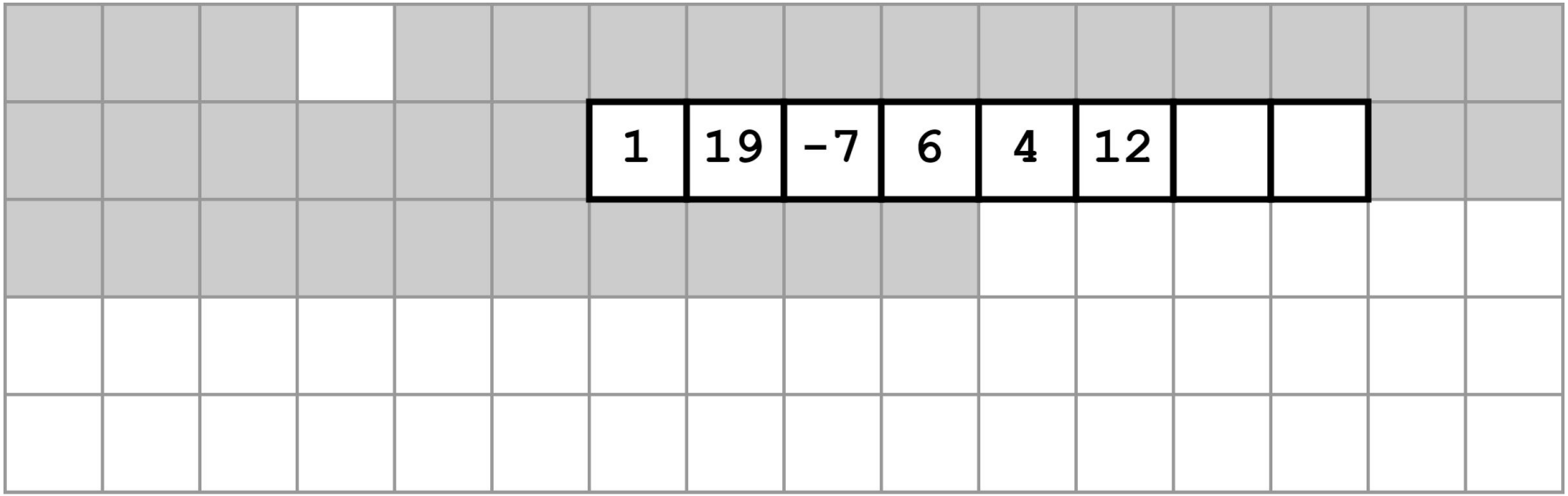
Program Memory

```
arr[4] = 4;
```



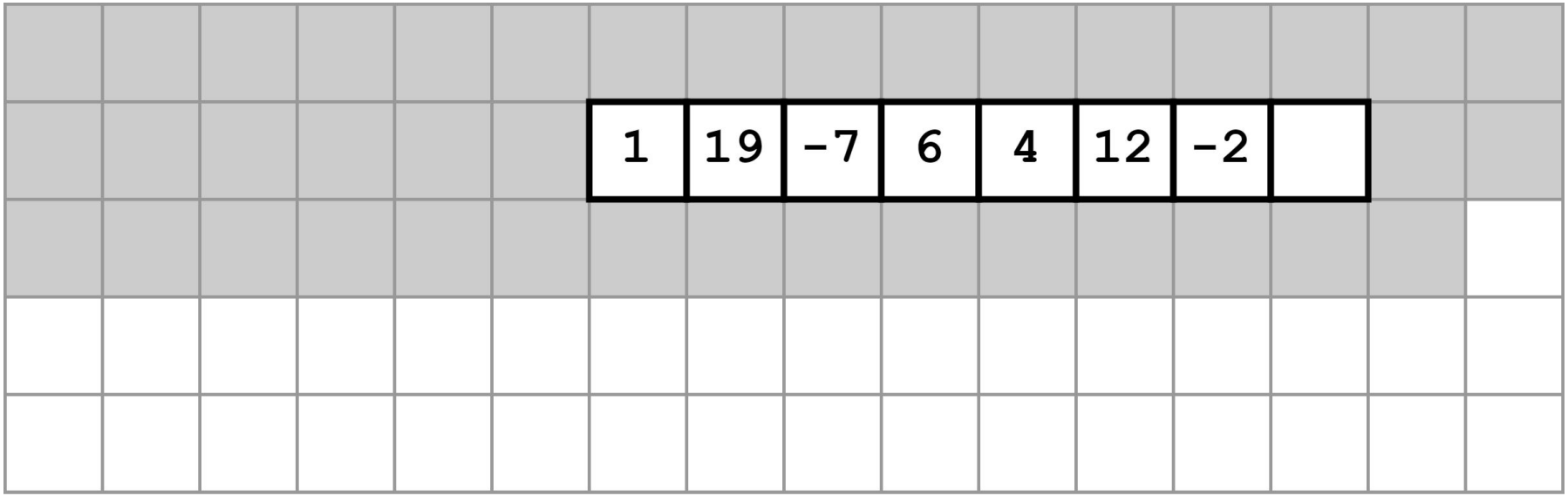
Program Memory

```
arr[5] = 12;
```



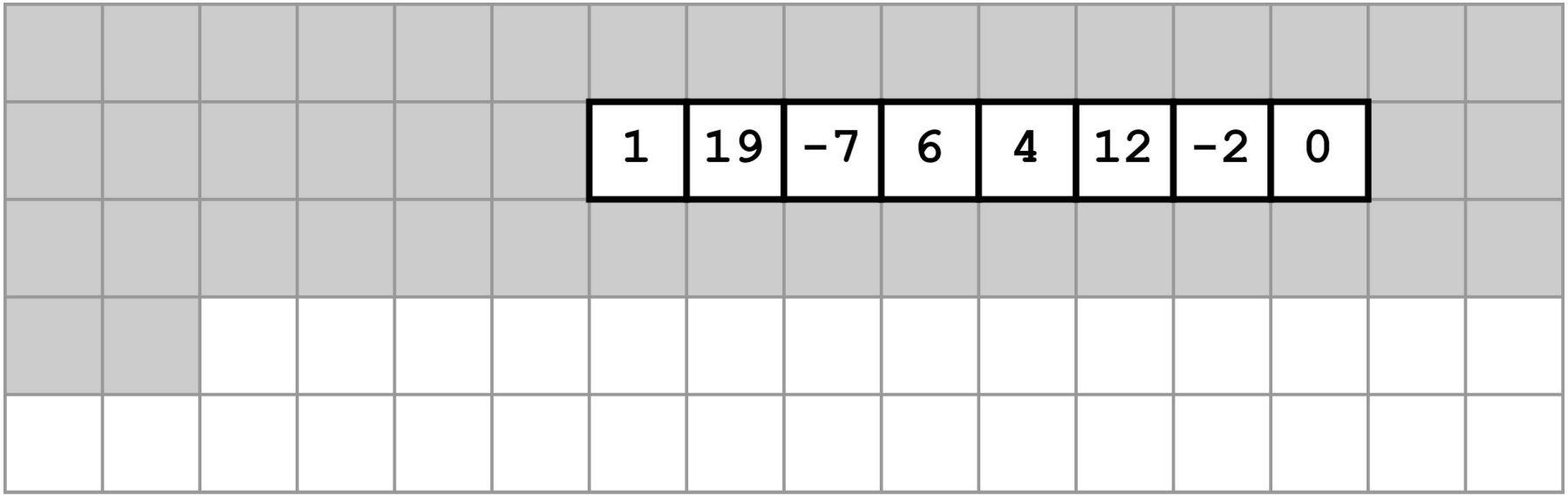
Program Memory

```
arr[6] = -2;
```



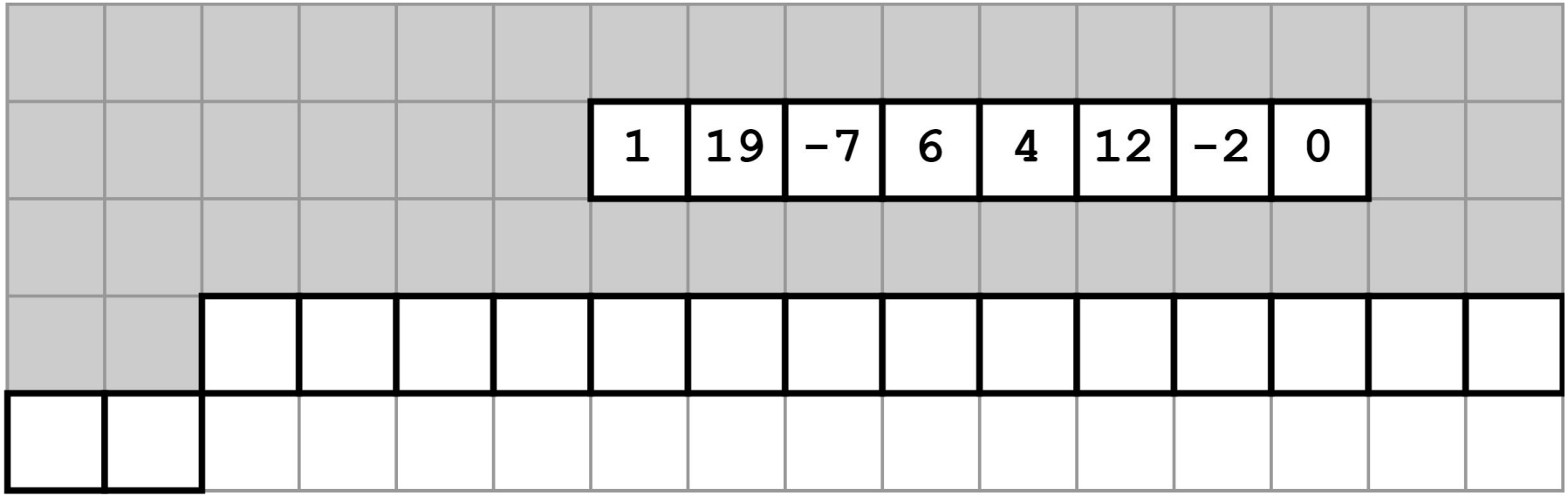
Program Memory

```
arr[7] = 0;
```

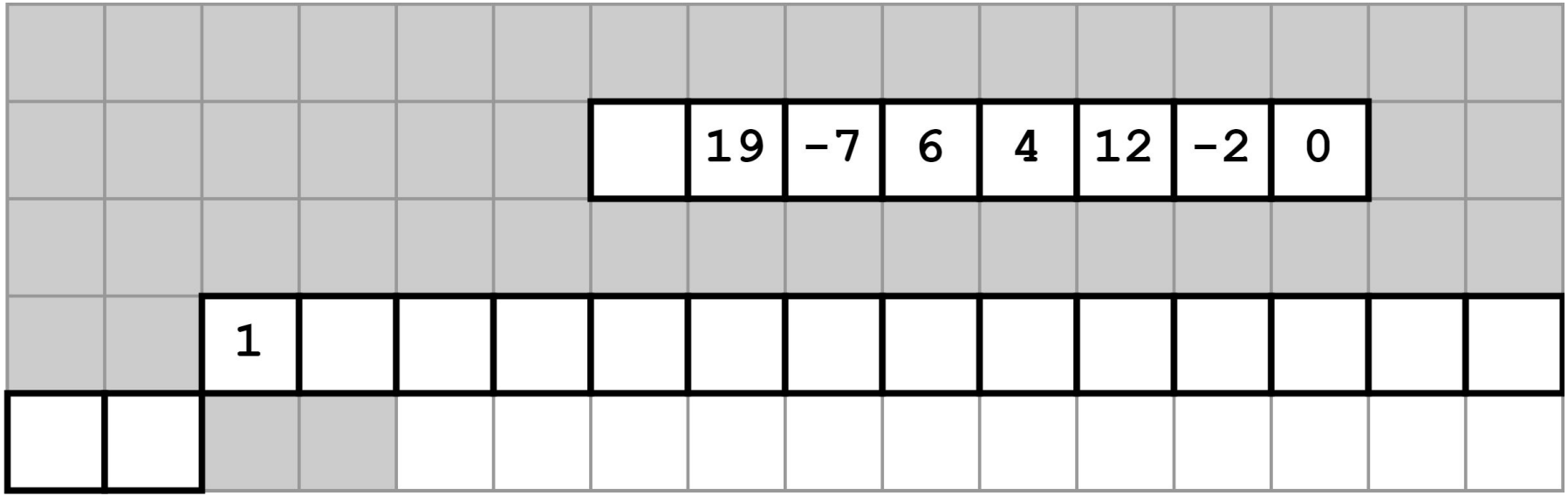


Program Memory

```
Integer[] temp = new Integer[16];
```

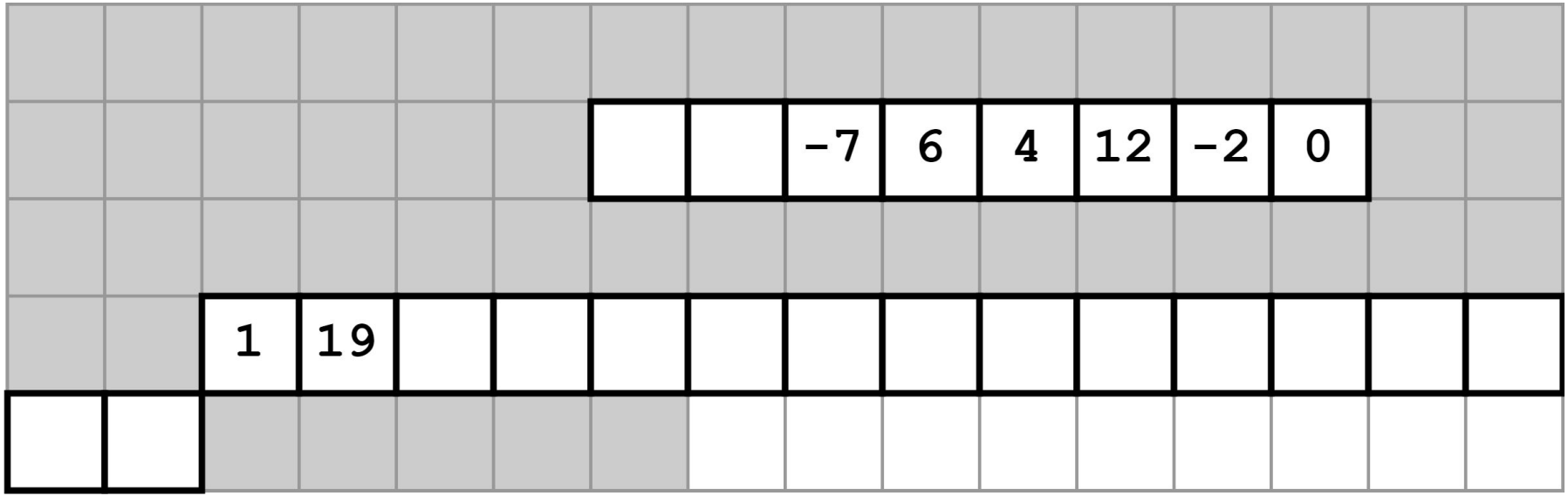


```
temp[0] = arr[0];
```



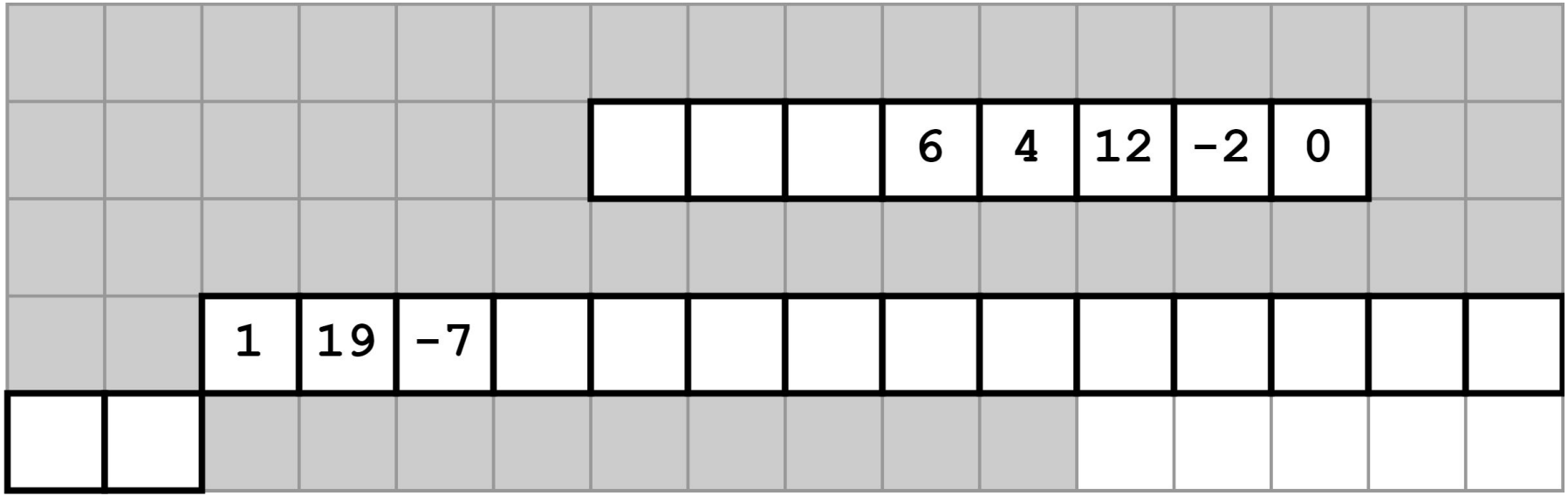
Program Memory

```
temp[1] = arr[1];
```



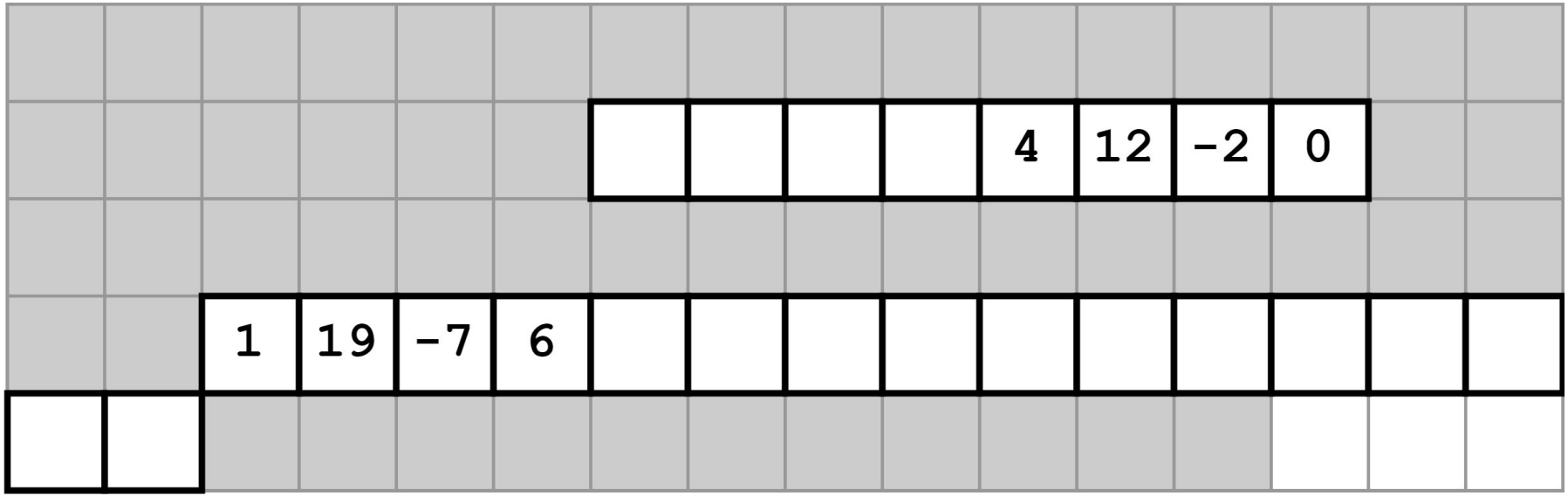
Program Memory

```
temp[2] = arr[2];
```



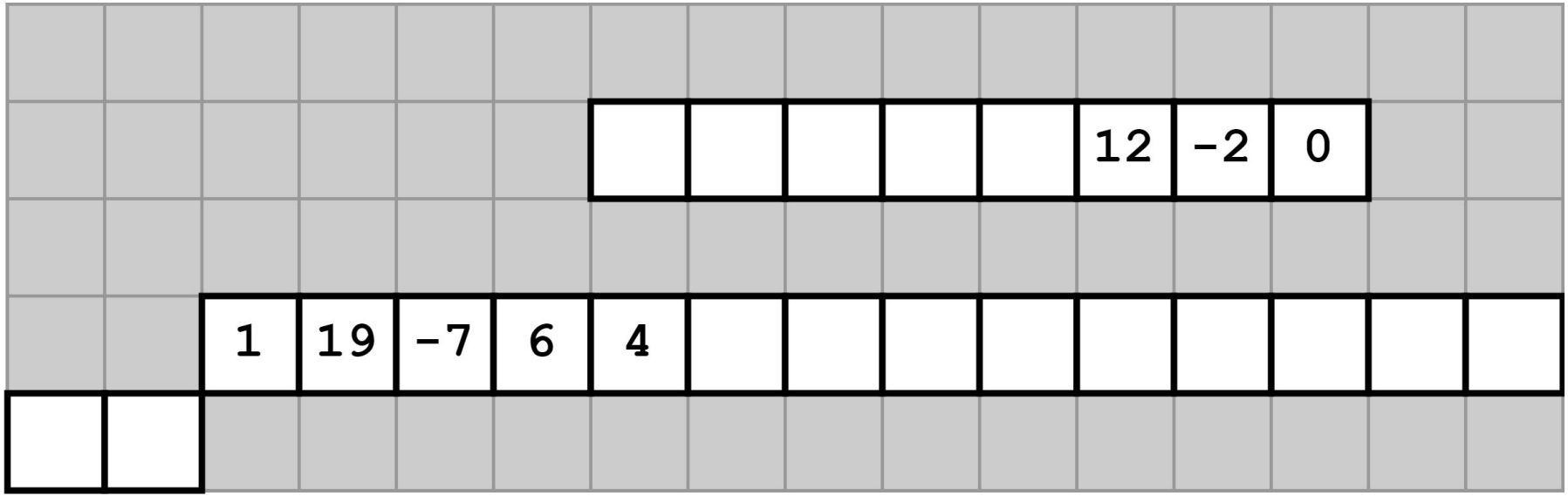
Program Memory

```
temp[3] = arr[3];
```



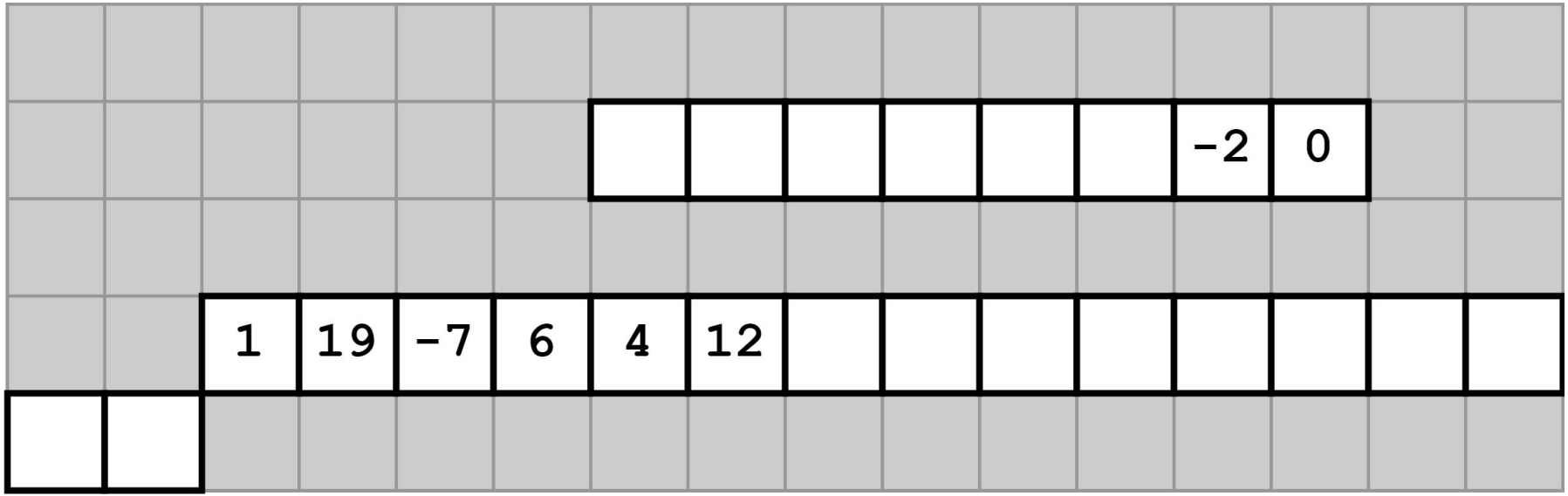
Program Memory

```
temp[4] = arr[4];
```



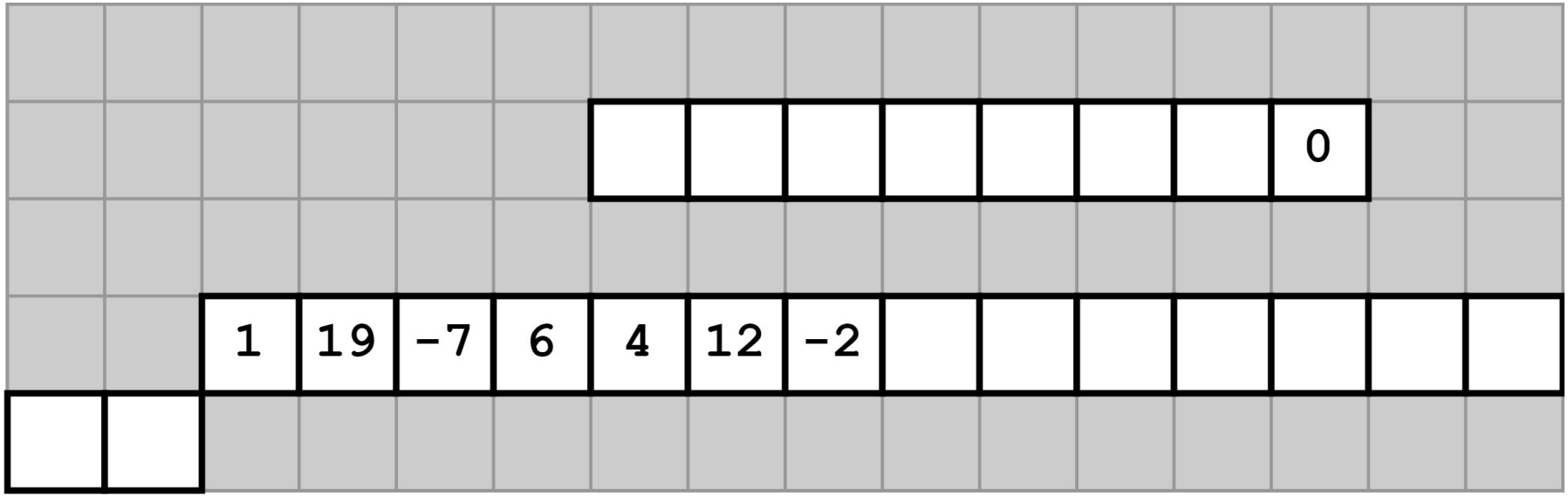
Program Memory

```
temp[5] = arr[5];
```



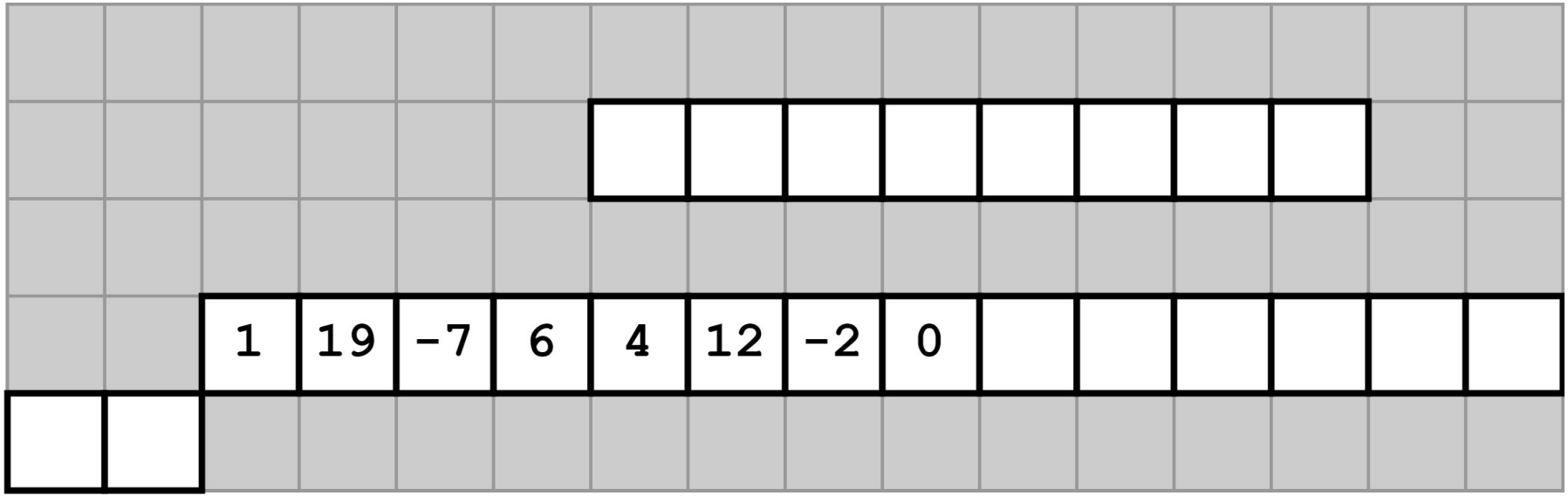
Program Memory

```
temp[6] = arr[6];
```



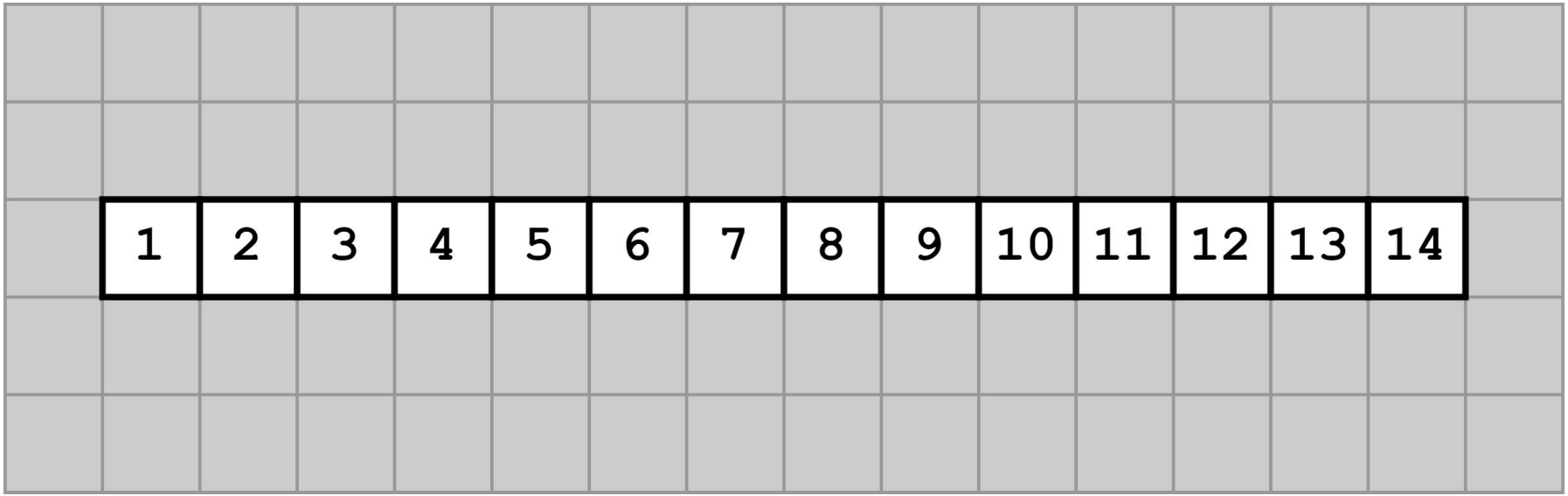
Program Memory

```
temp[7] = arr[7];
```

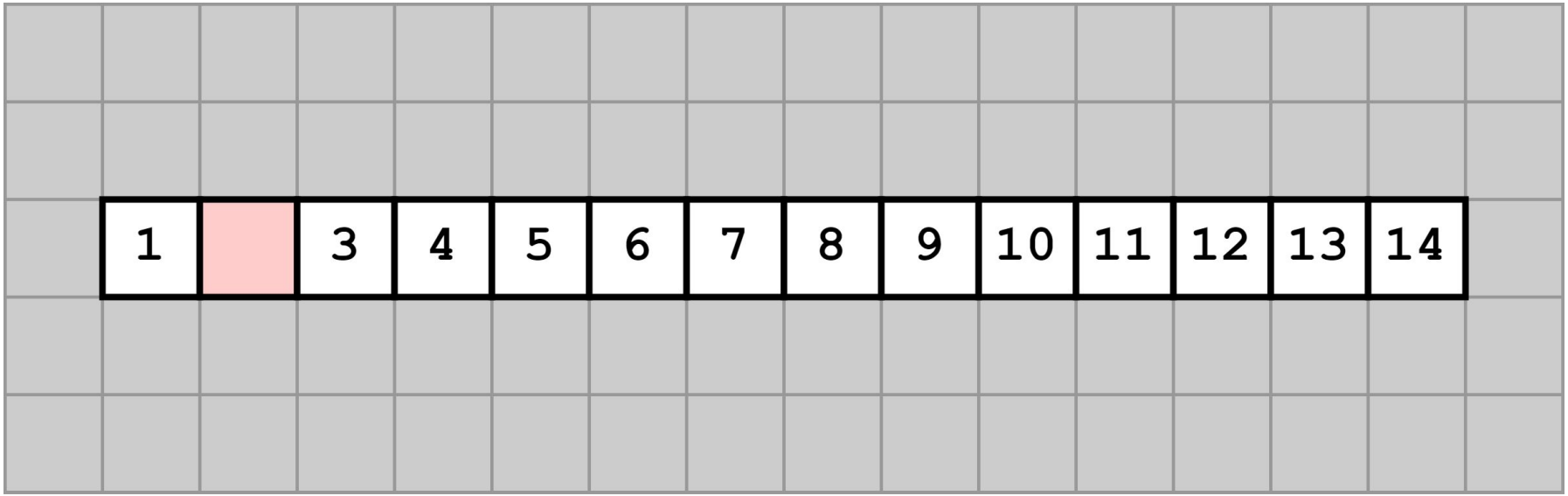


Program Memory

A new array issue

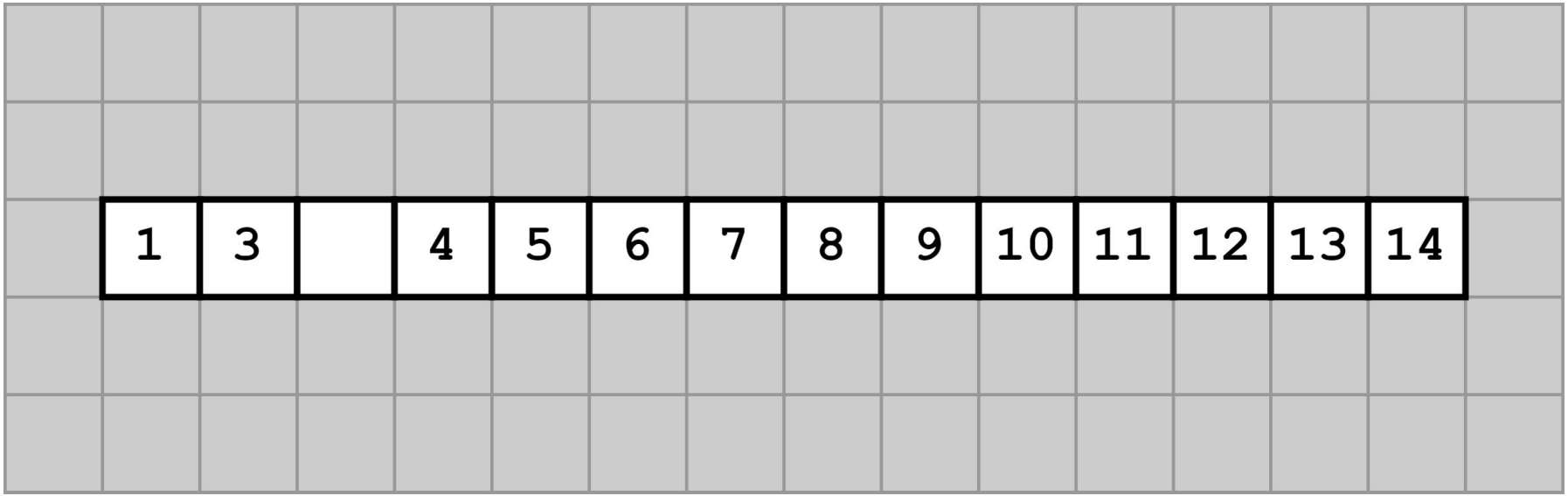


```
arr[1] = null;
```



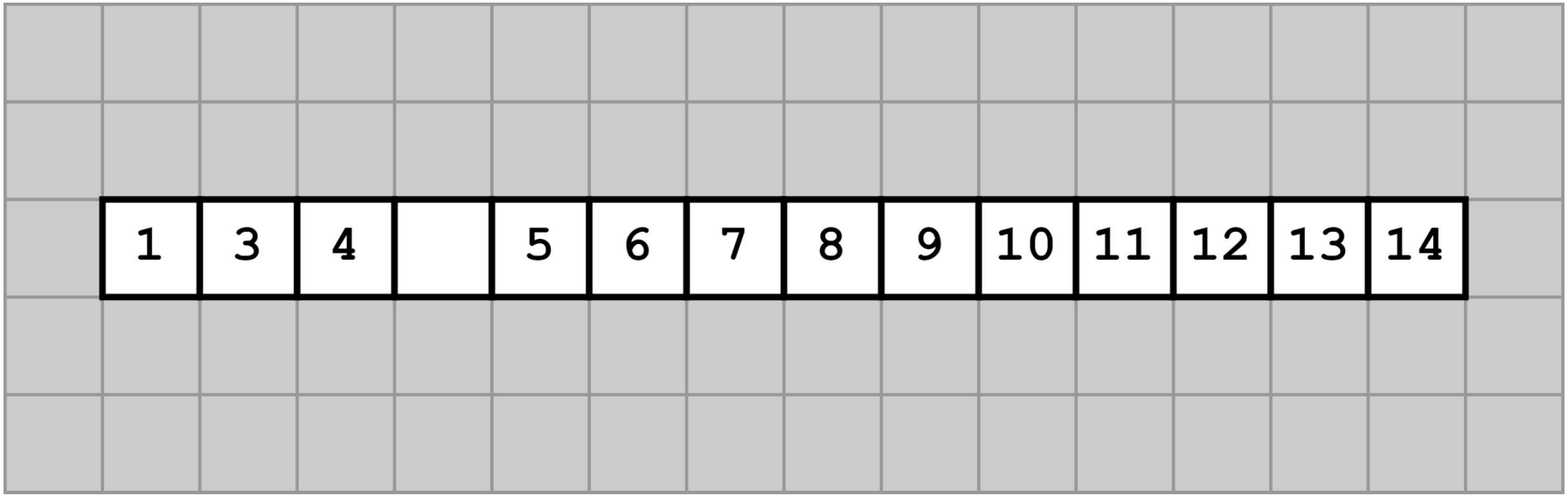
Program Memory

```
arr[1] = arr[2]; arr[2] = null;
```



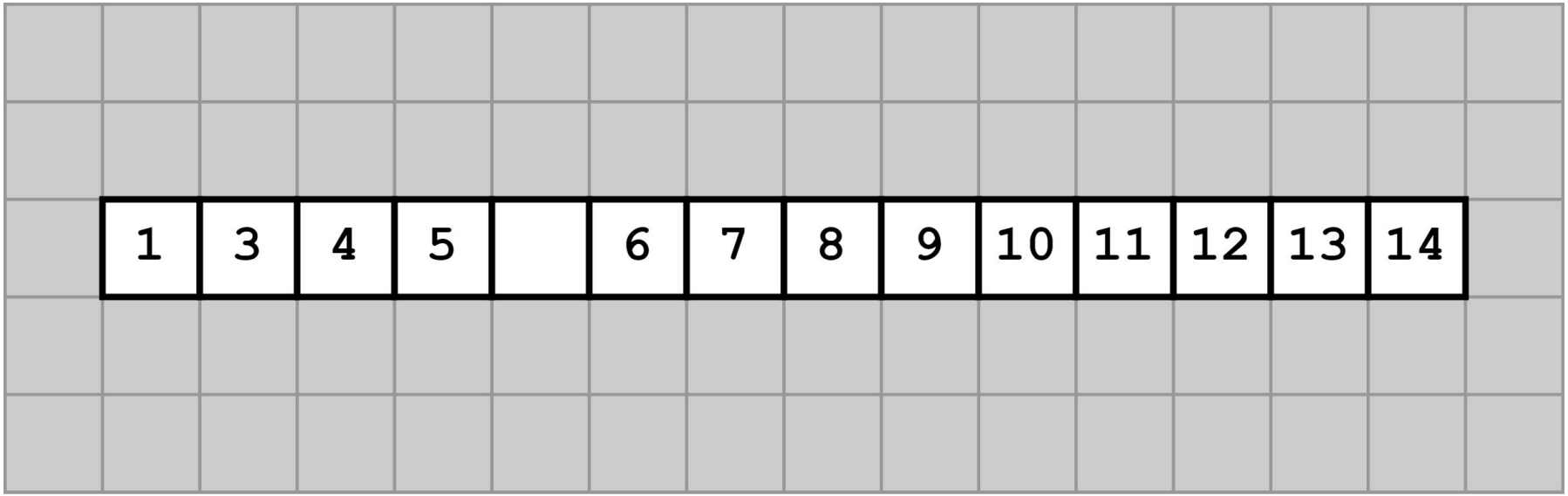
Program Memory

```
arr[2] = arr[3]; arr[3] = null;
```



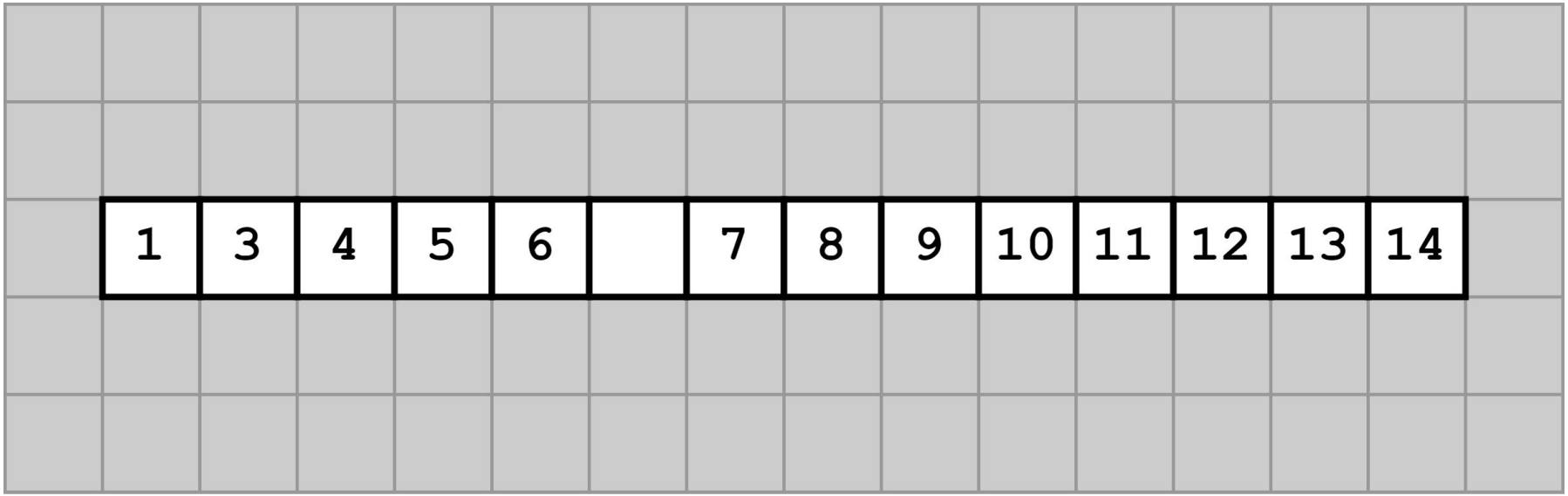
Program Memory

```
arr[3] = arr[4]; arr[4] = null;
```



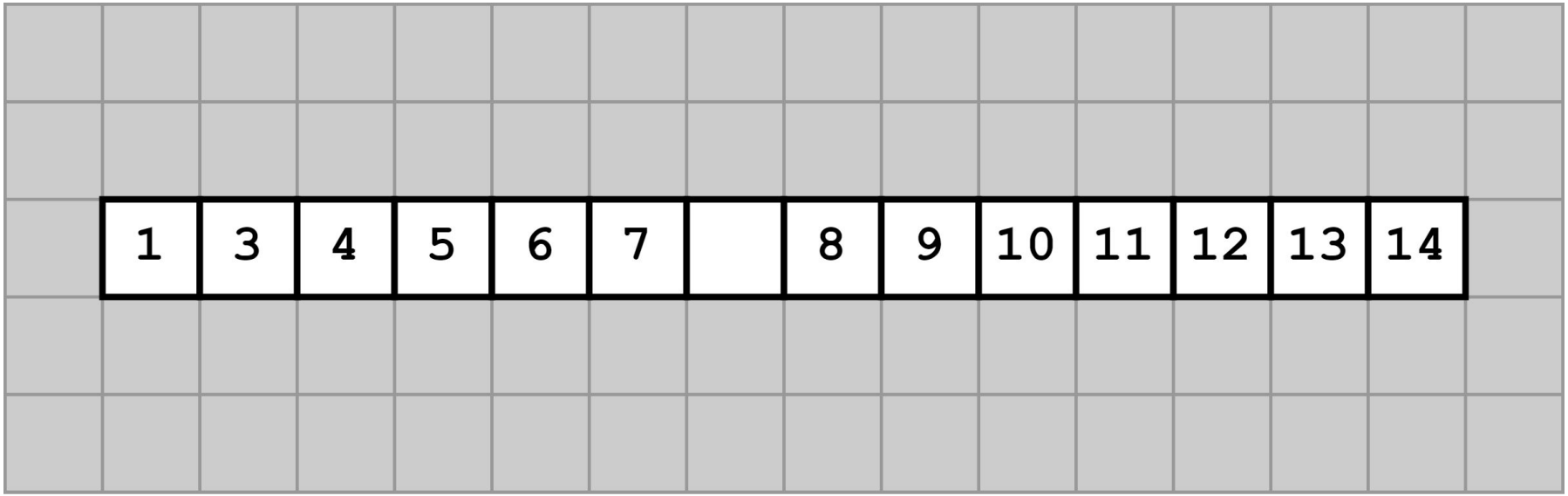
Program Memory

```
arr[4] = arr[5]; arr[5] = null;
```



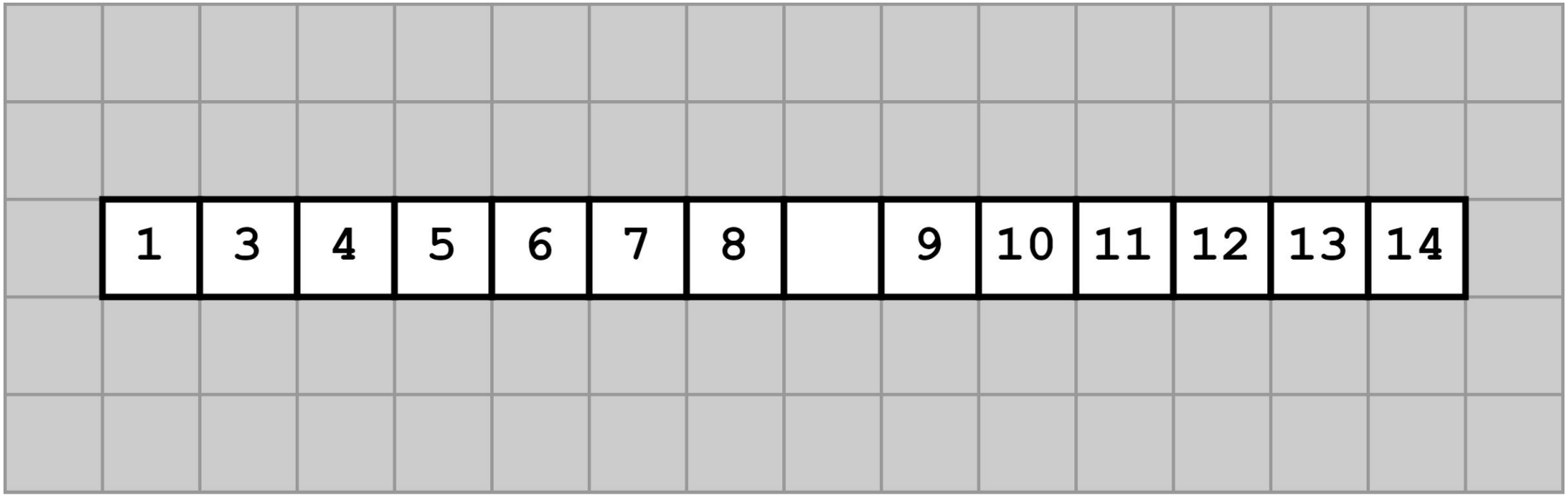
Program Memory

```
arr[5] = arr[6]; arr[6] = null;
```



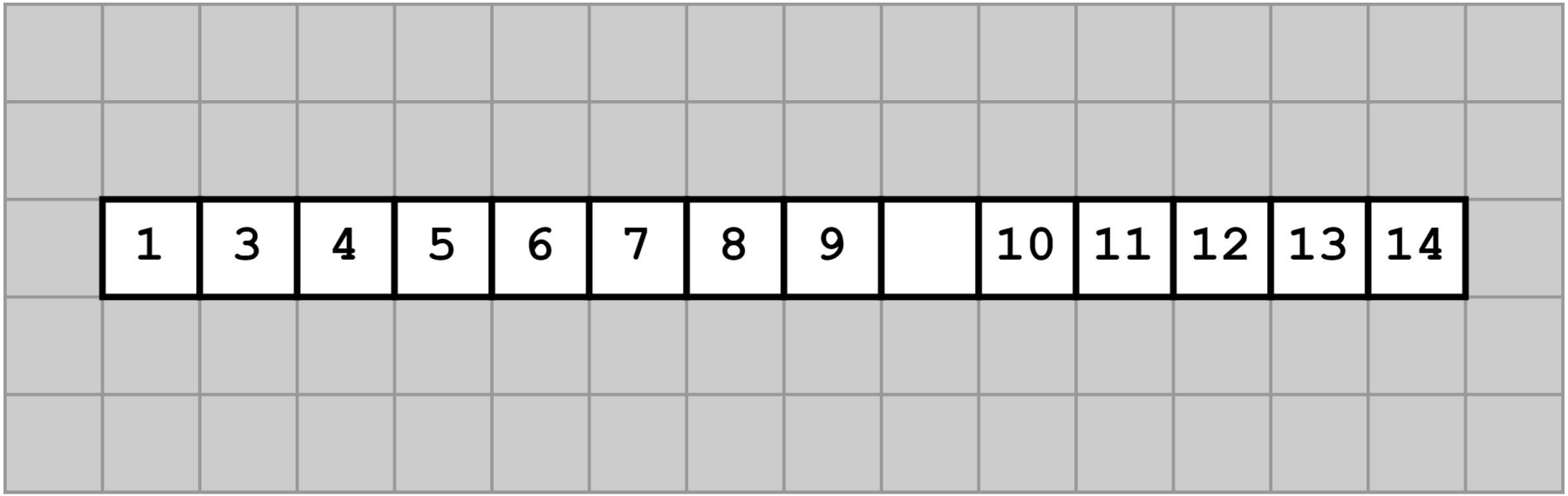
Program Memory

```
arr[6] = arr[7]; arr[7] = null;
```



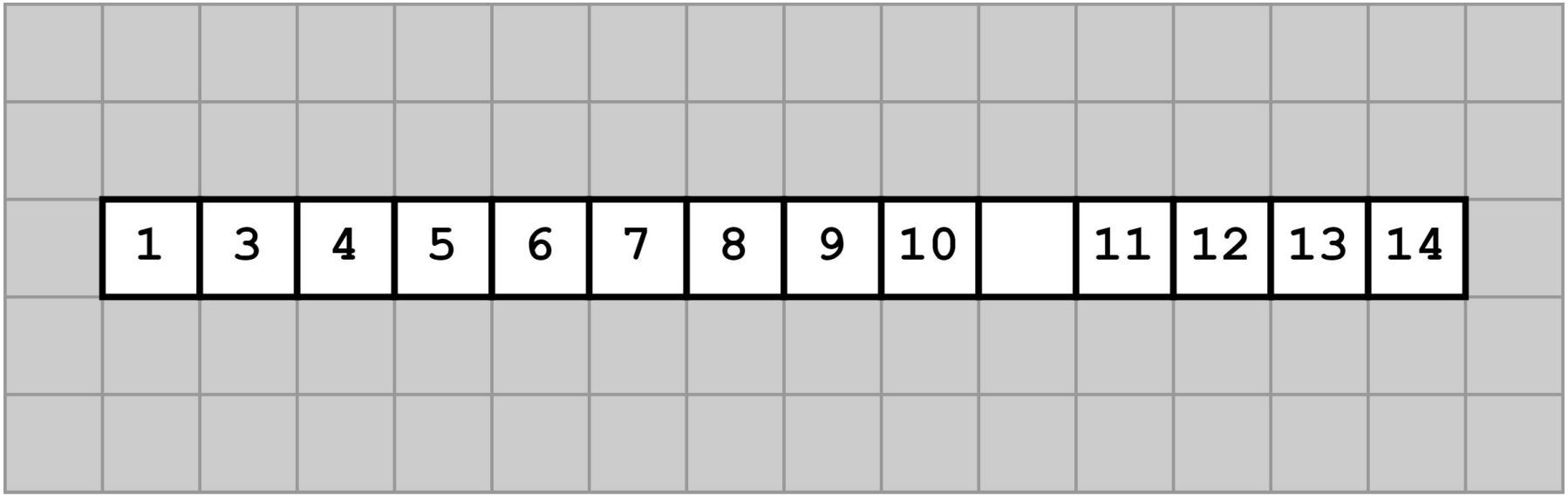
Program Memory

```
arr[7] = arr[8]; arr[8] = null;
```



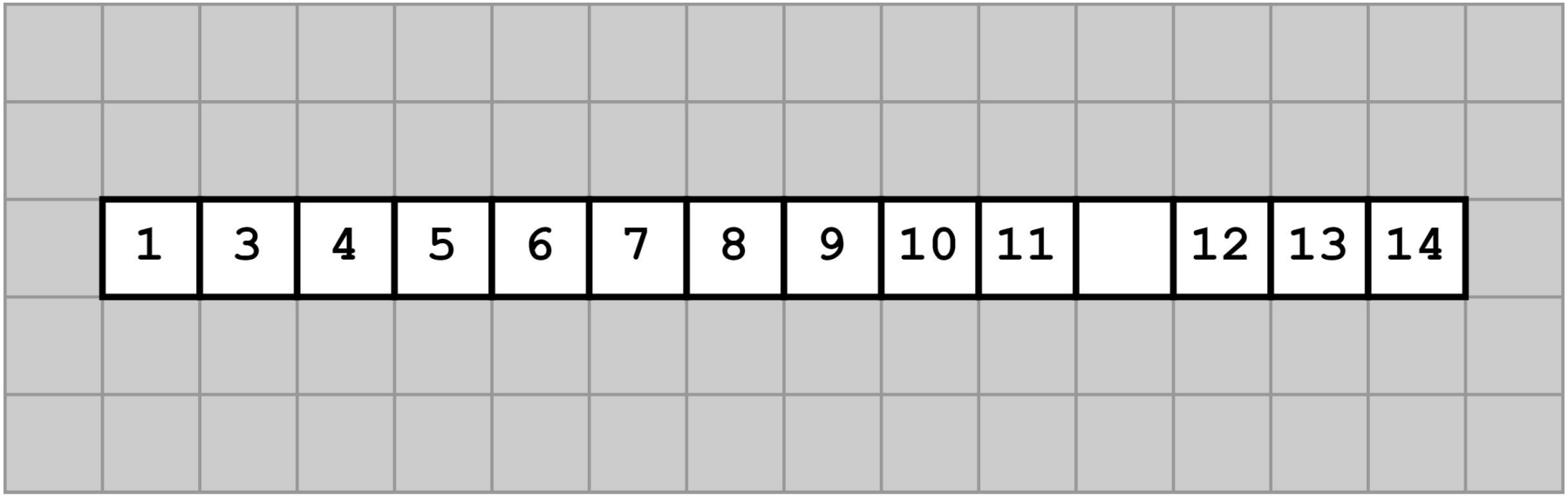
Program Memory

```
arr[8] = arr[9]; arr[9] = null;
```



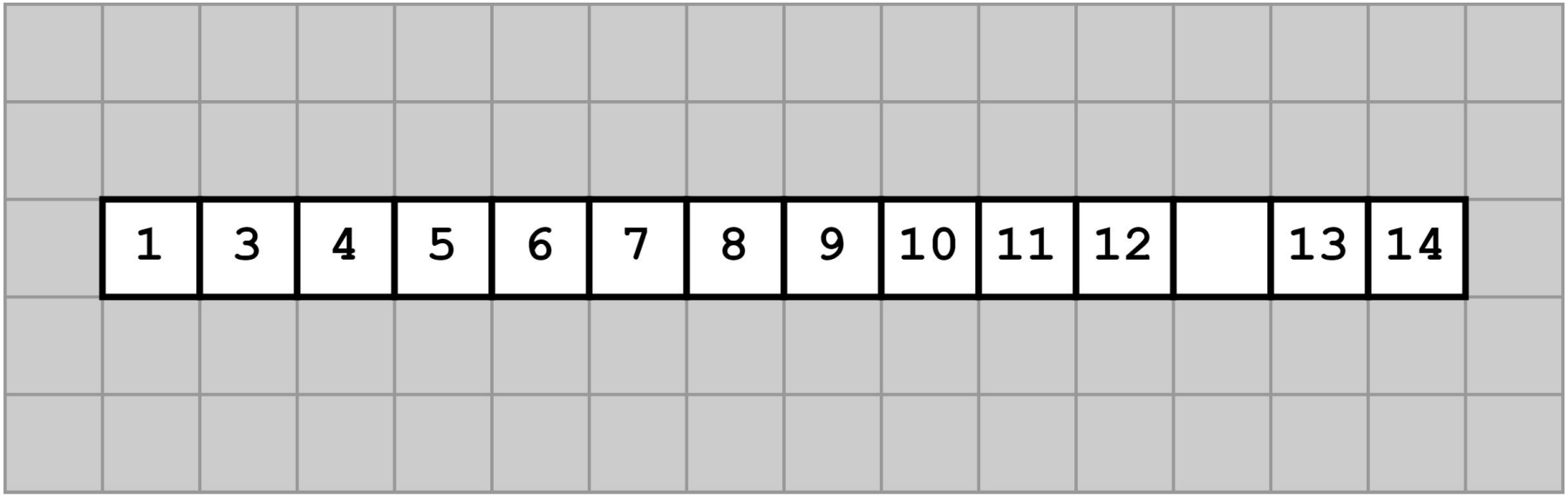
Program Memory

```
arr[9] = arr[10]; arr[10] = null;
```



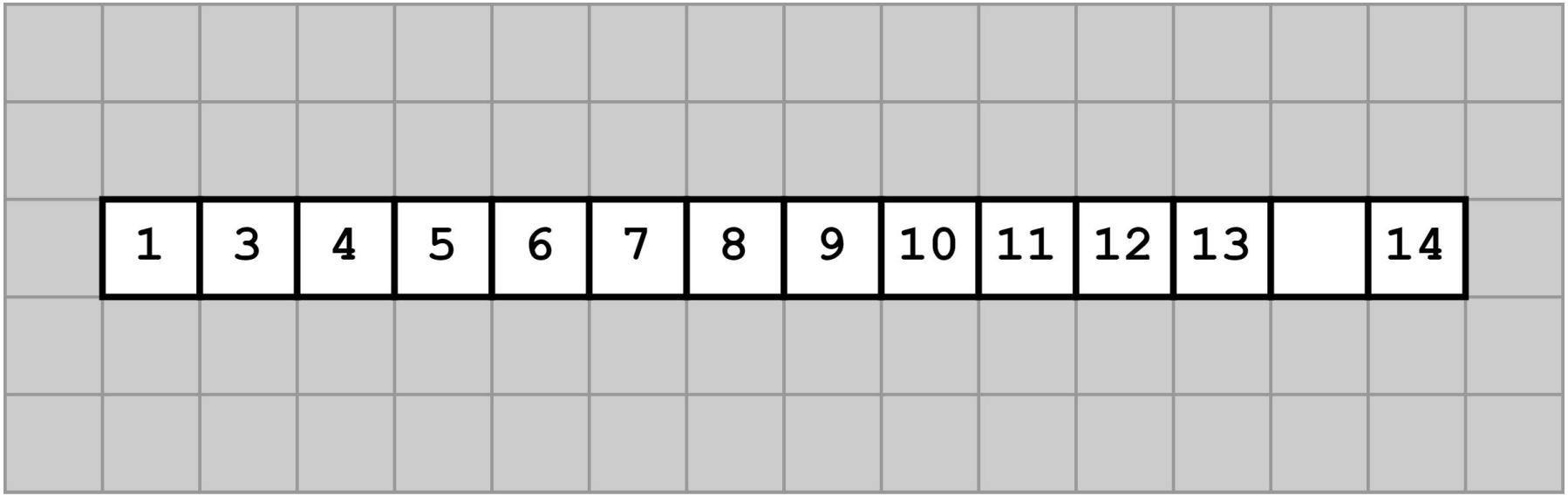
Program Memory

```
arr[10] = arr[11]; arr[11] = null;
```



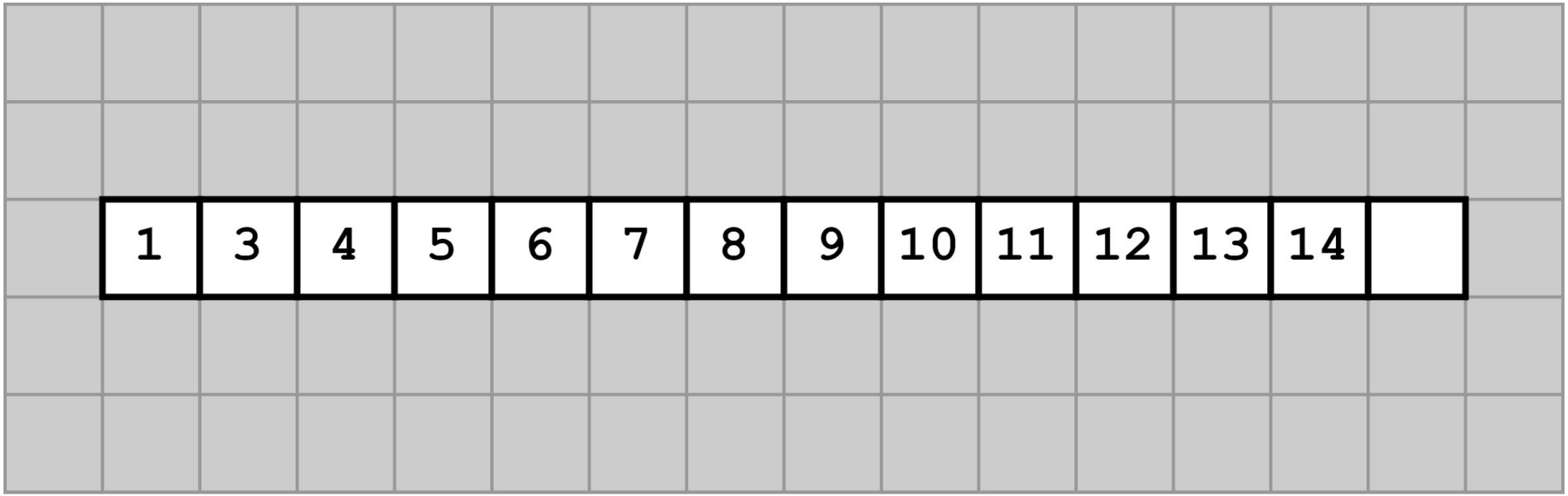
Program Memory

```
arr[11] = arr[12]; arr[12] = null;
```



Program Memory

```
arr[12] = arr[13]; arr[13] = null;
```



Program Memory

Downsides of Arrays

- Cannot dynamically increase their size; O(n) to change size
- O(n) to remove element
- O(n) to add element

Alternatives?