Implementing Abstractions

Getting Storage Space

- How do the **`Vector`**, **`Stack`**, **`Queue`**, etc. get space to store all the elements that they hold?
- C++ code can request extra storage space as the program is running.
- This is called **dynamic memory allocation**.

Dynamic Allocation Demo

```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

numValues **7**

```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

numValues **7**

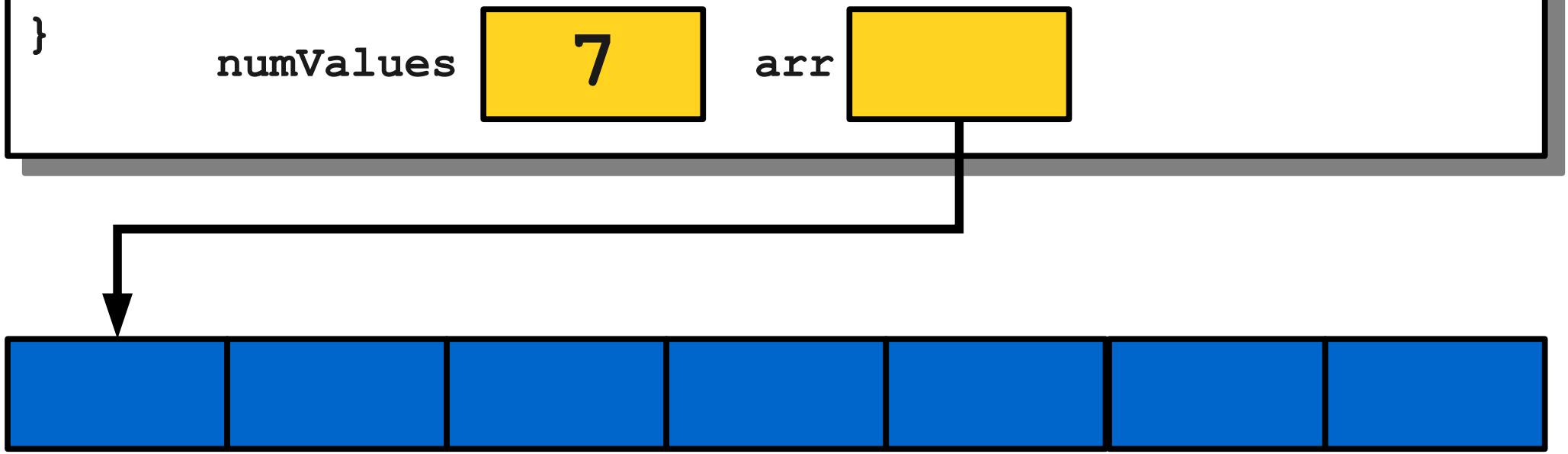
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

numValues | 7 | arr | |

```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

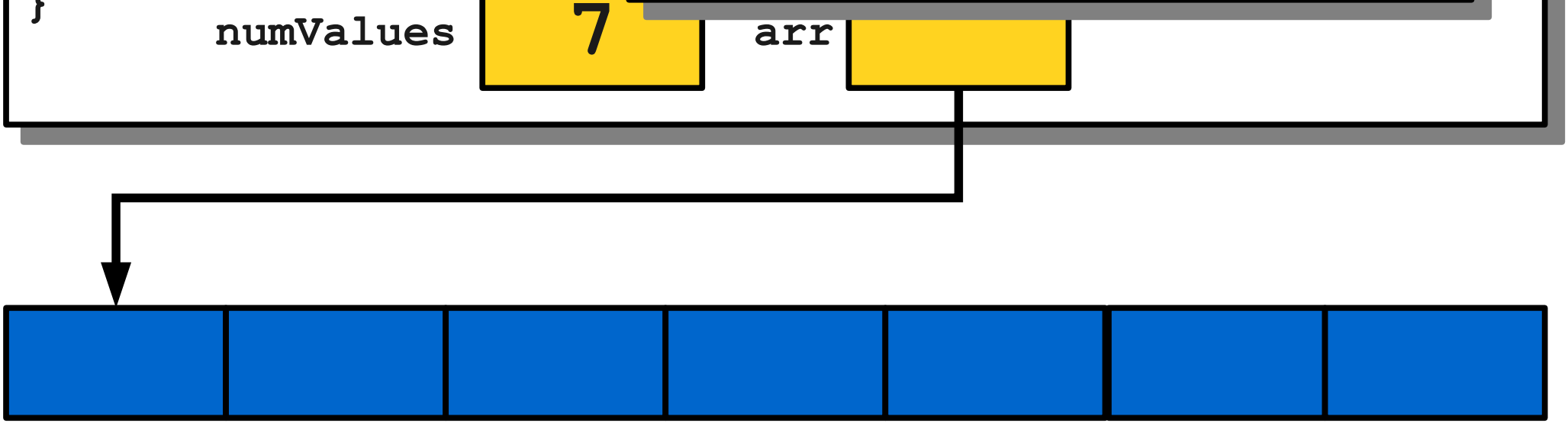
numValues 7 arr

```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

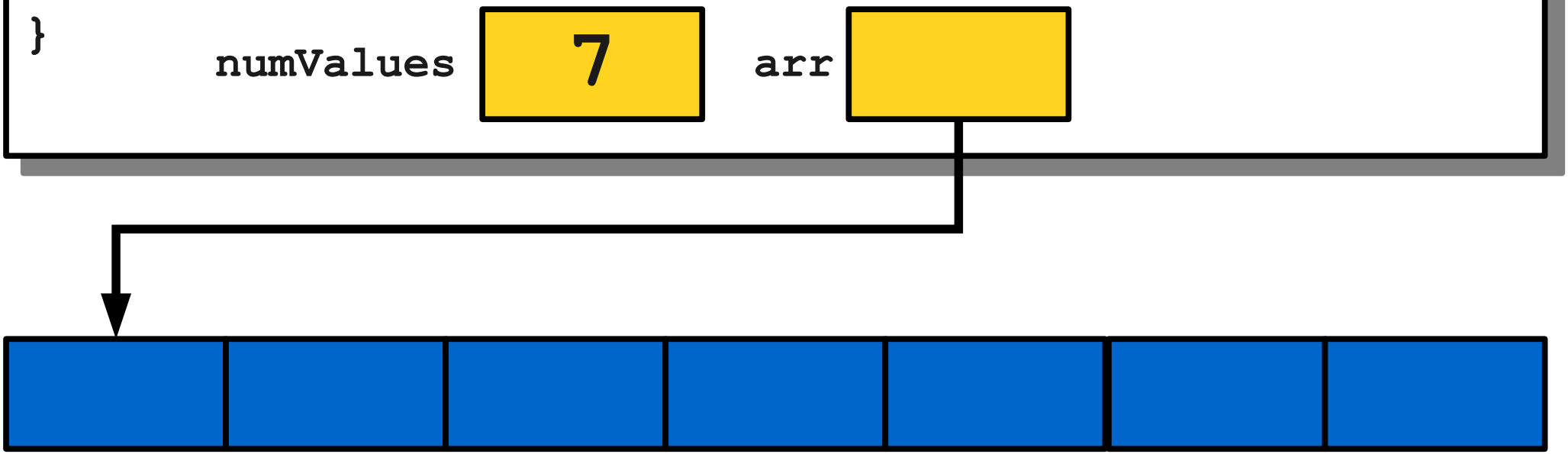


```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i
      cout << i << ":
   }
}
```

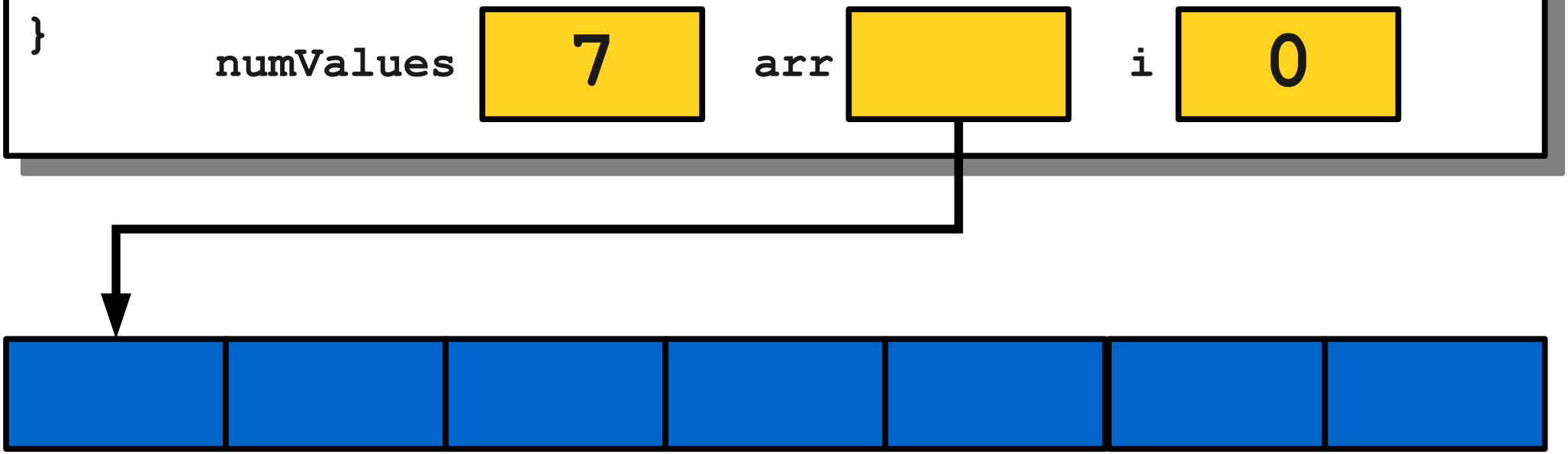
Because the variable **arr** points to the array, it is called a *pointer*.



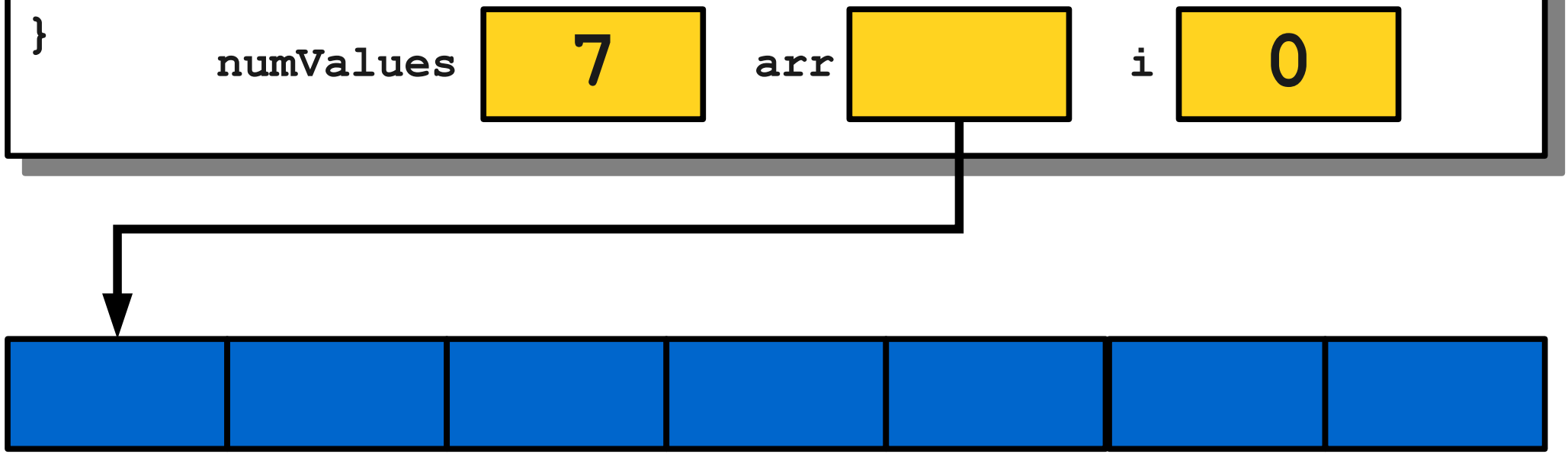
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



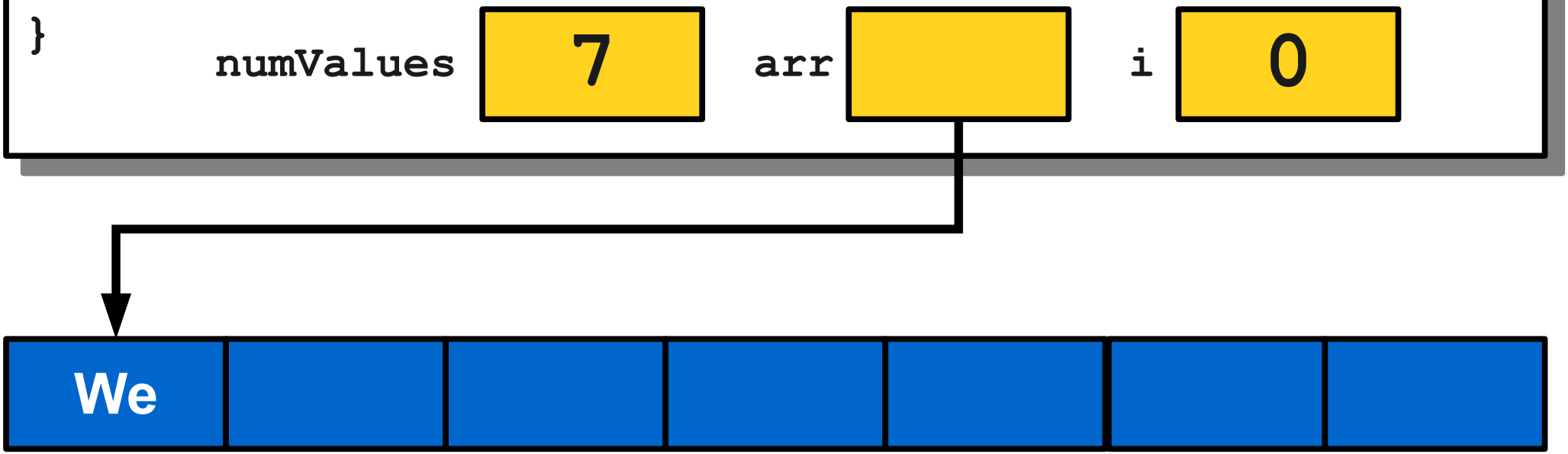
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



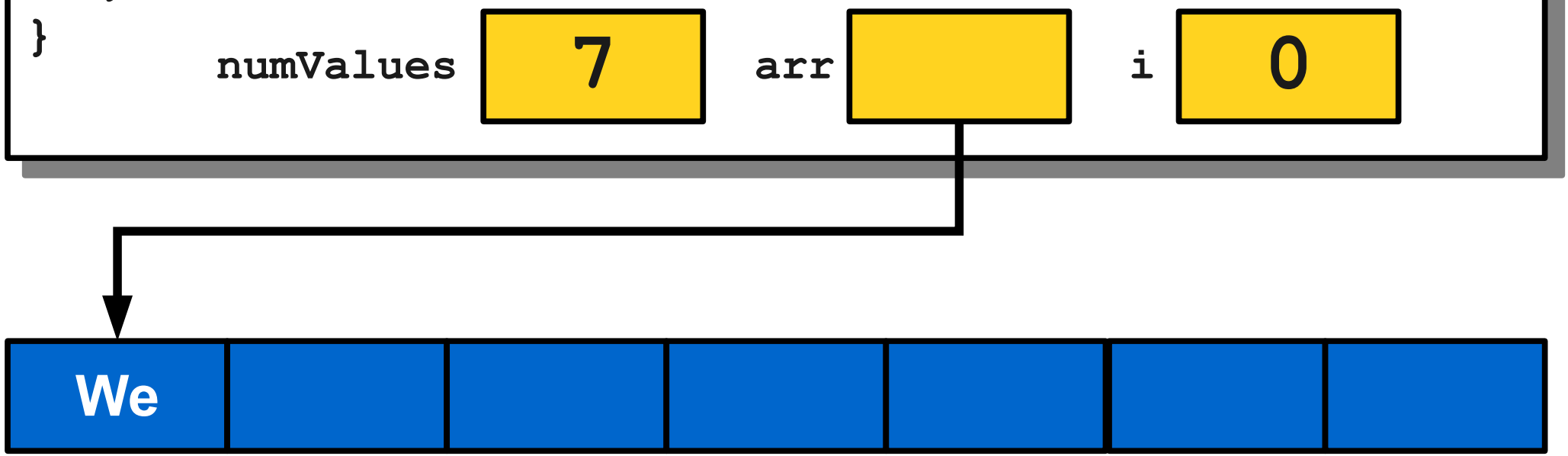
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



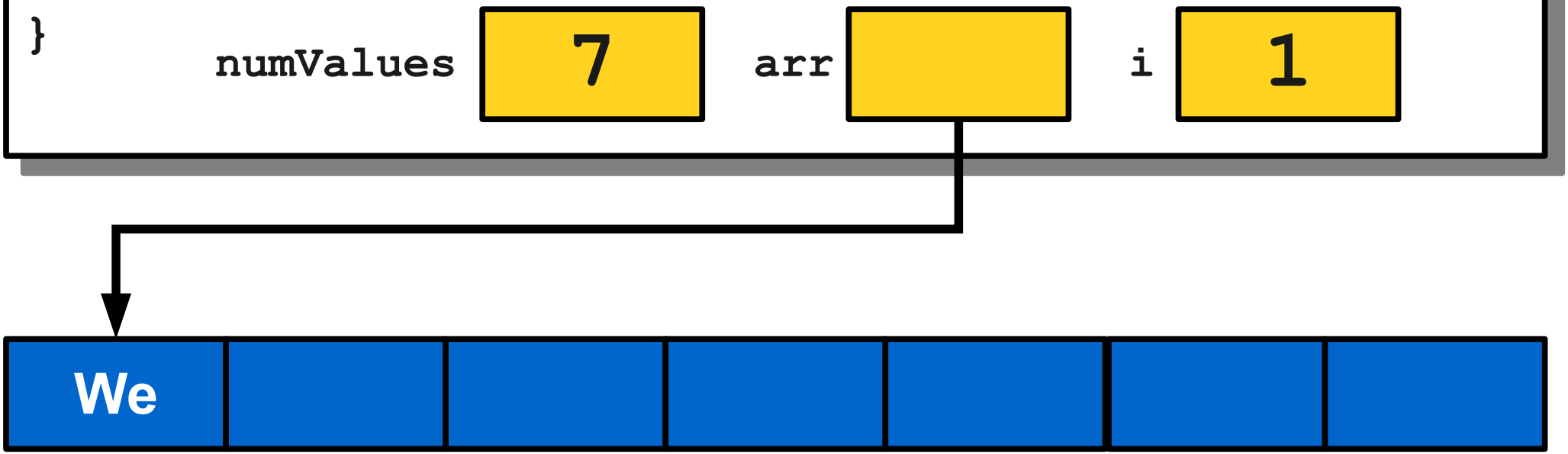
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



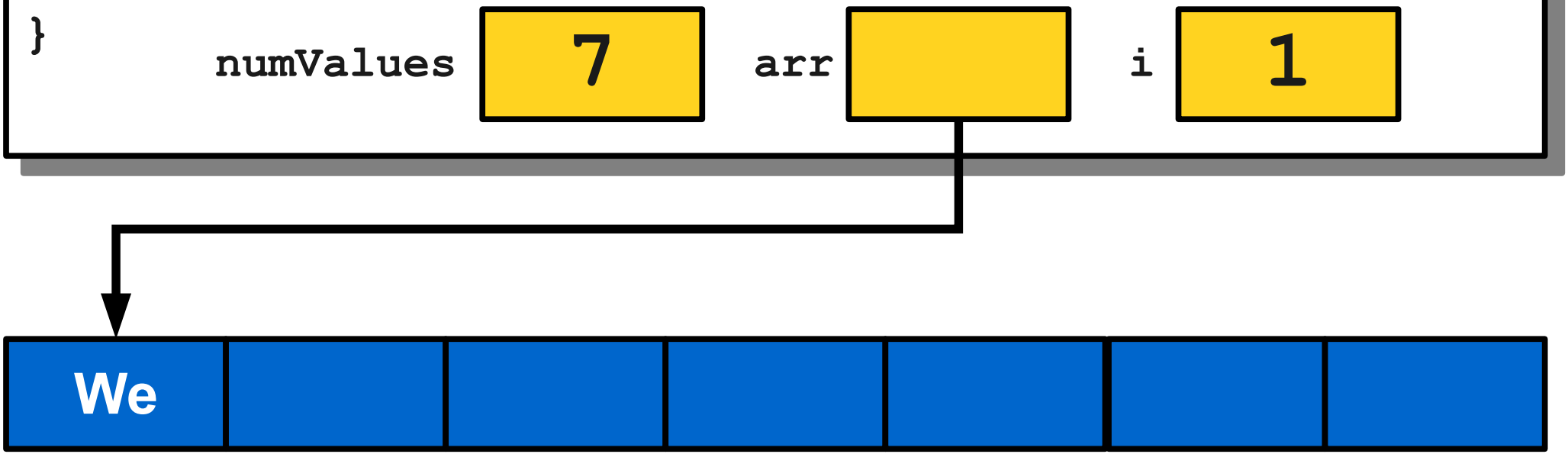
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



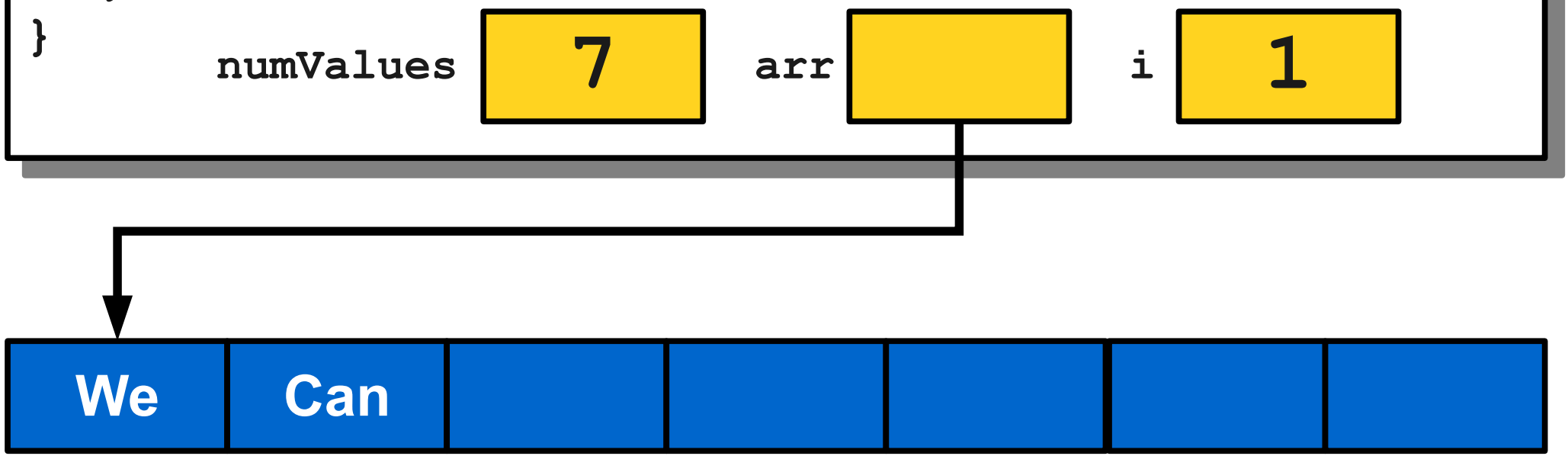
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



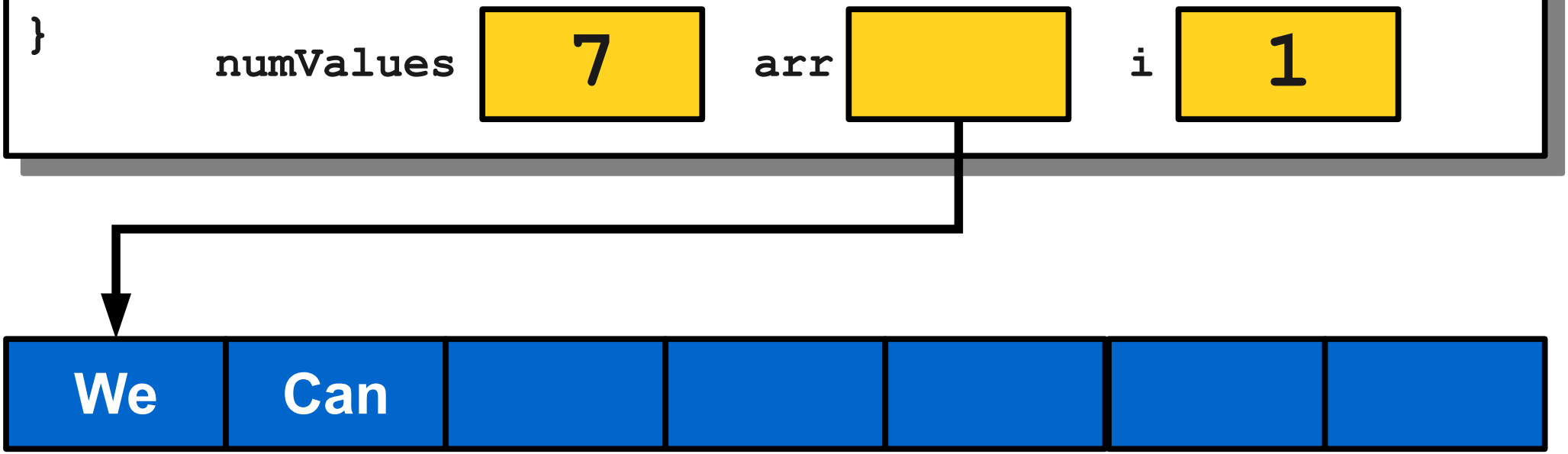
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



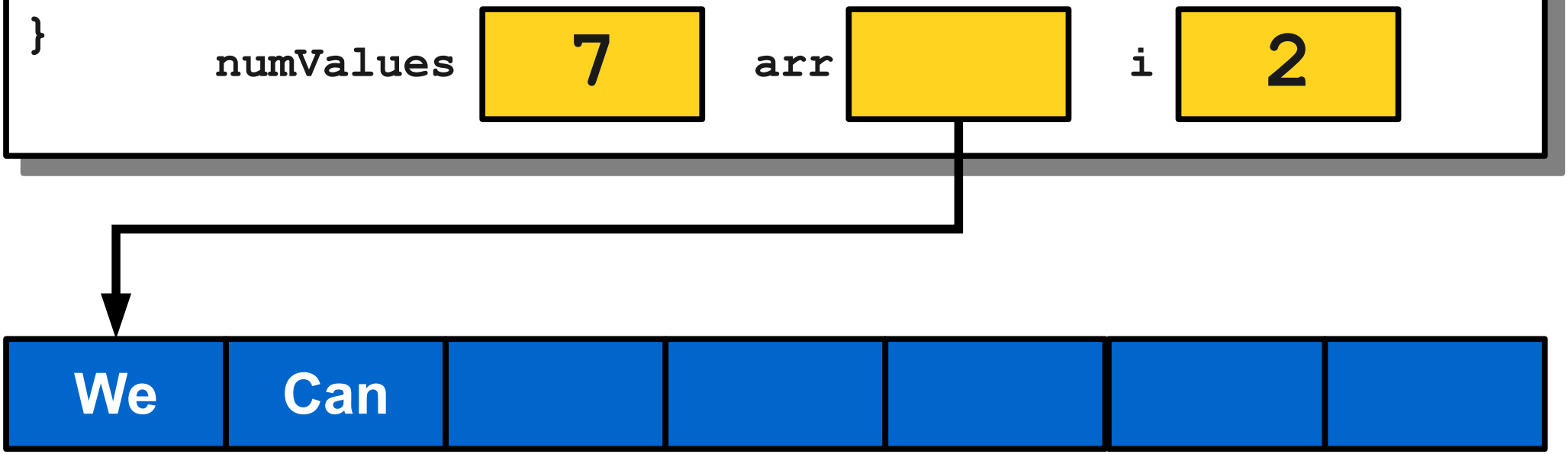
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



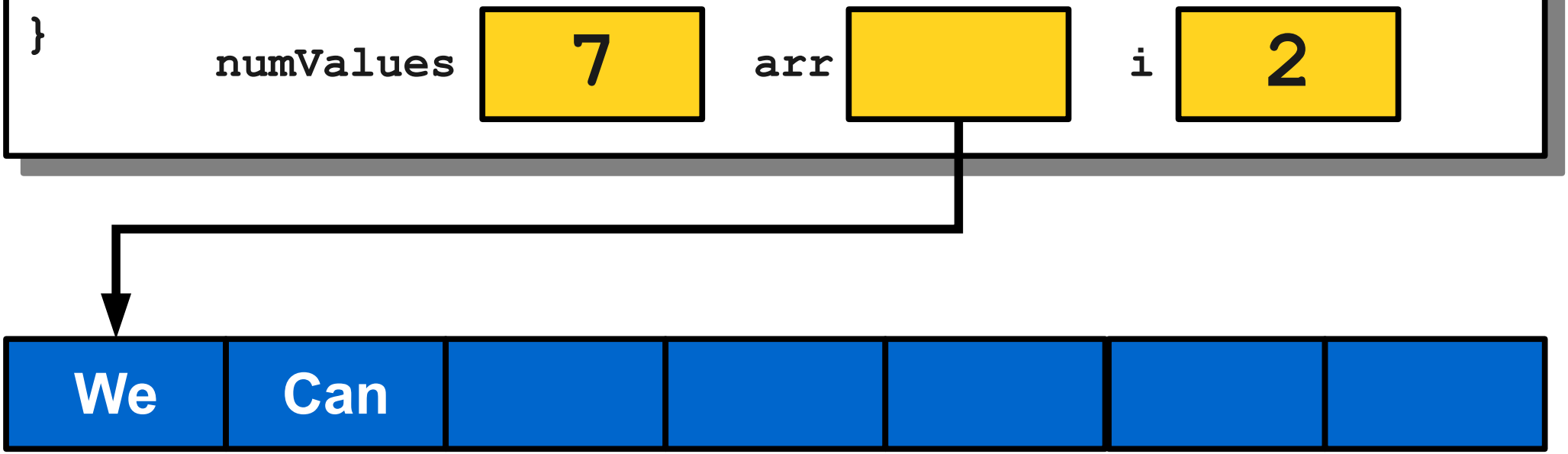
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



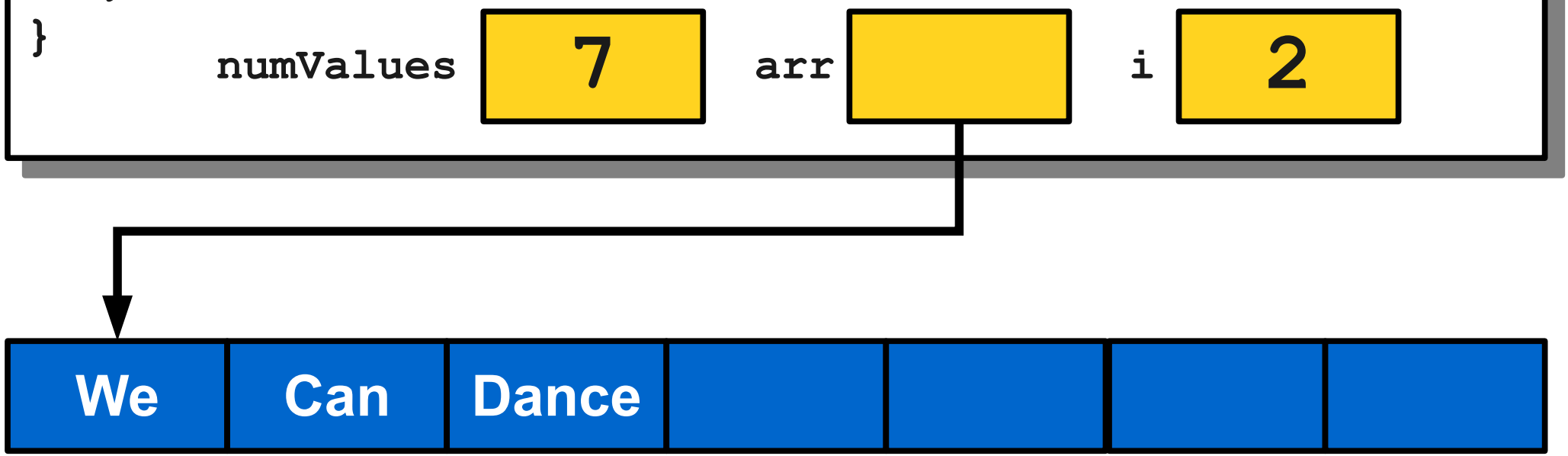
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



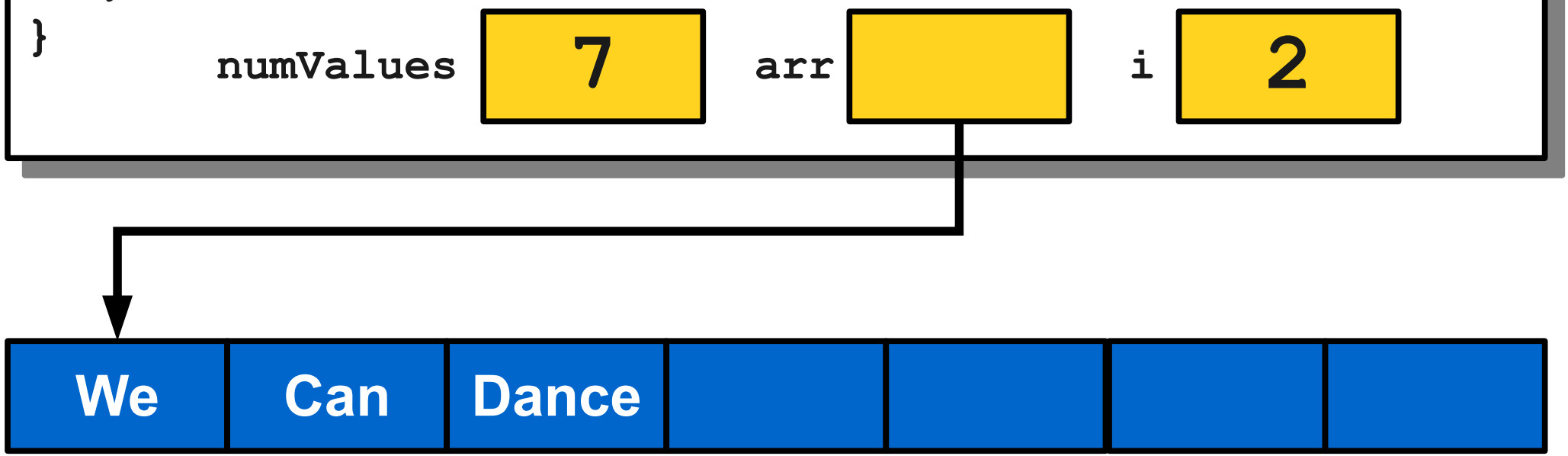
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



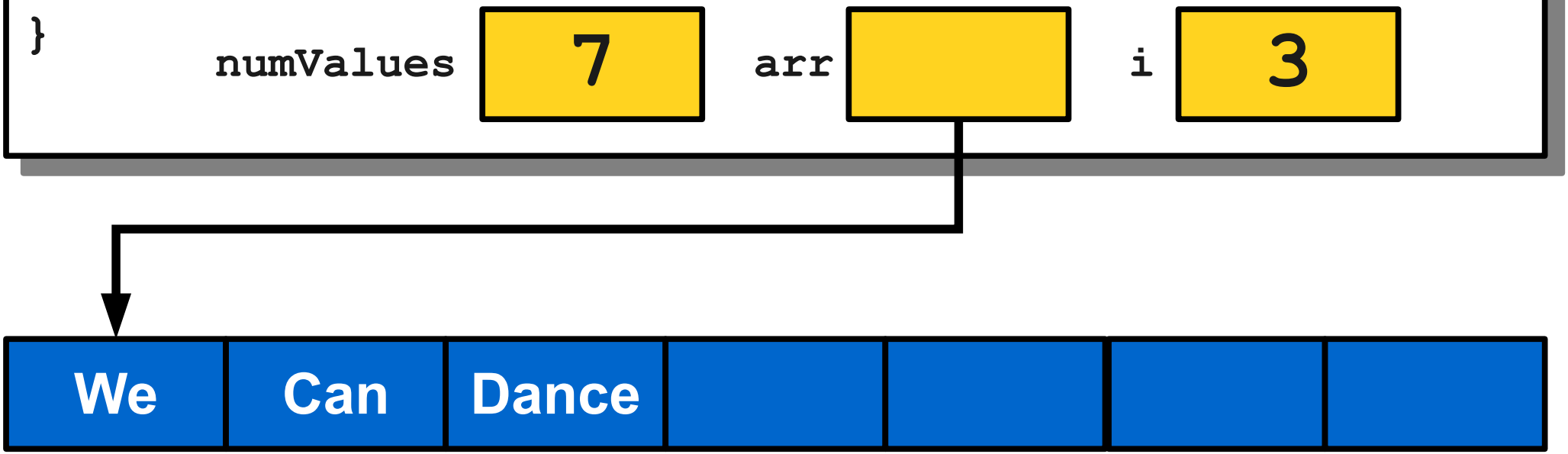
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



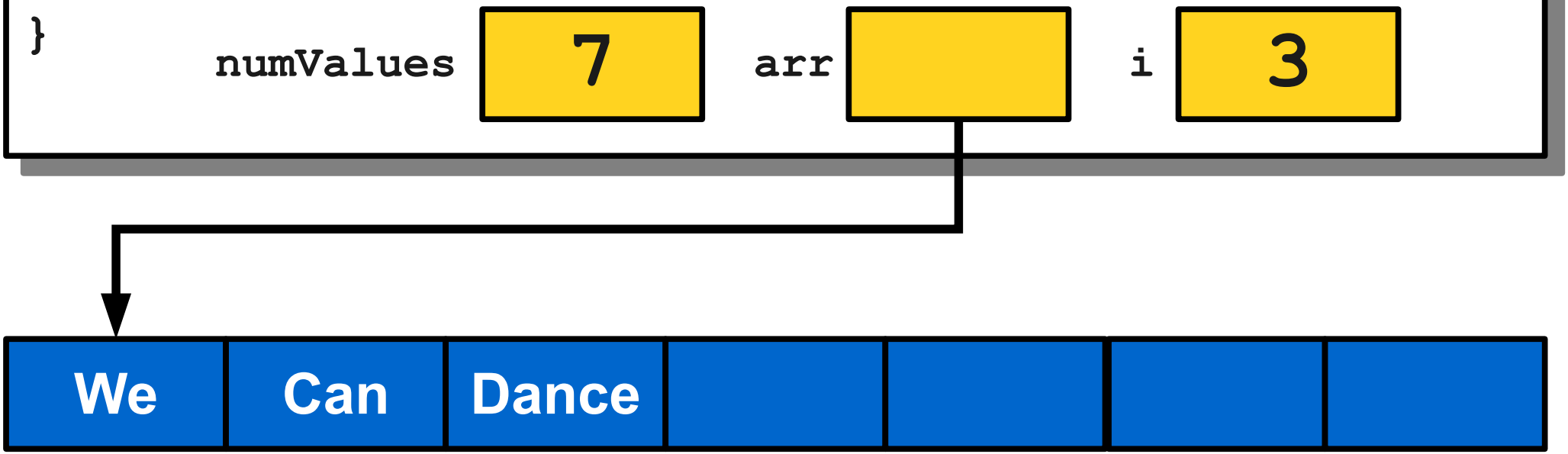
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



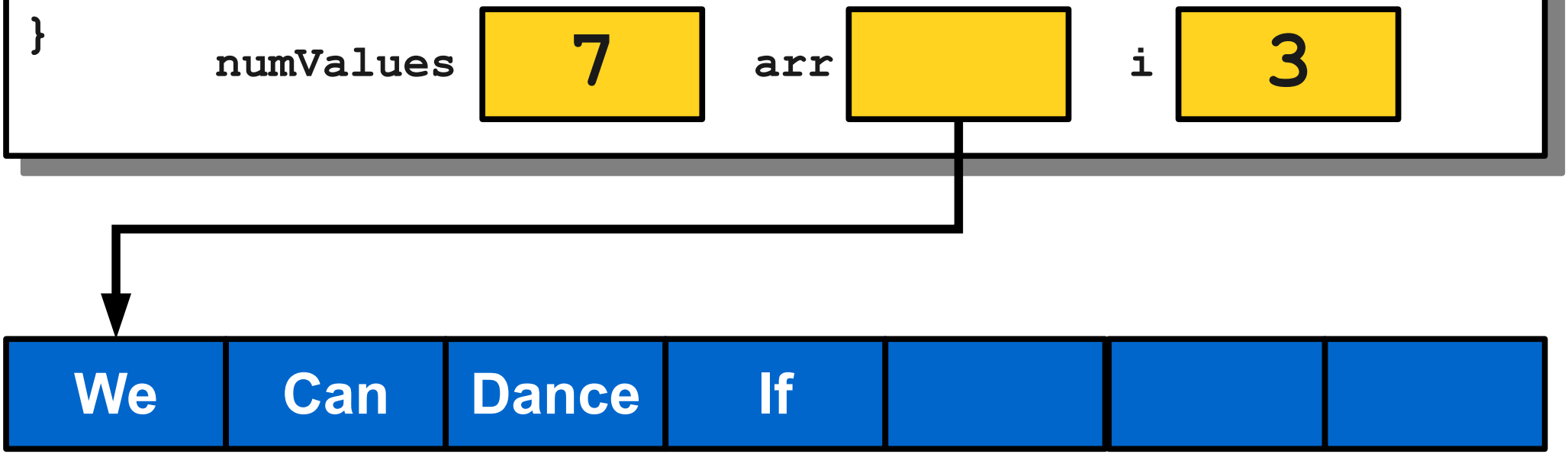
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



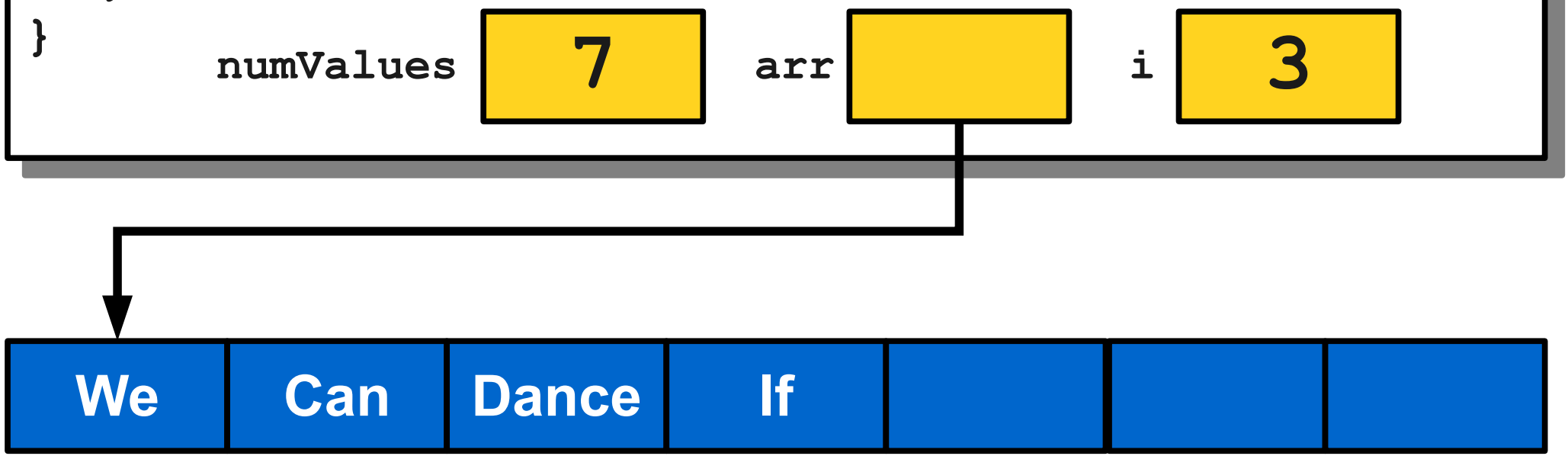
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



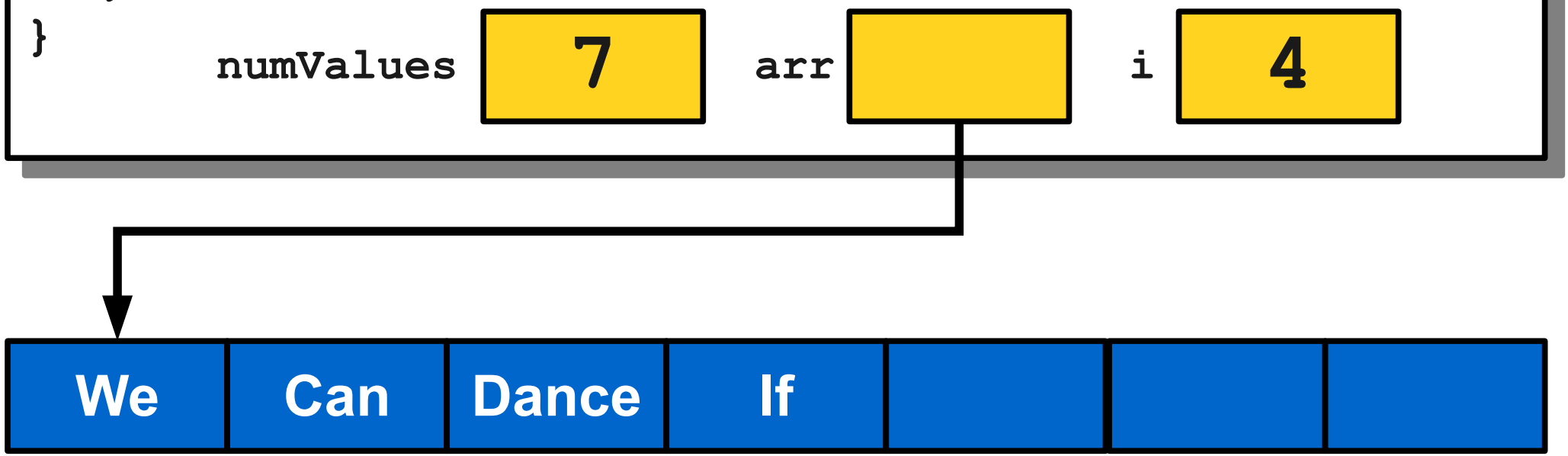
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



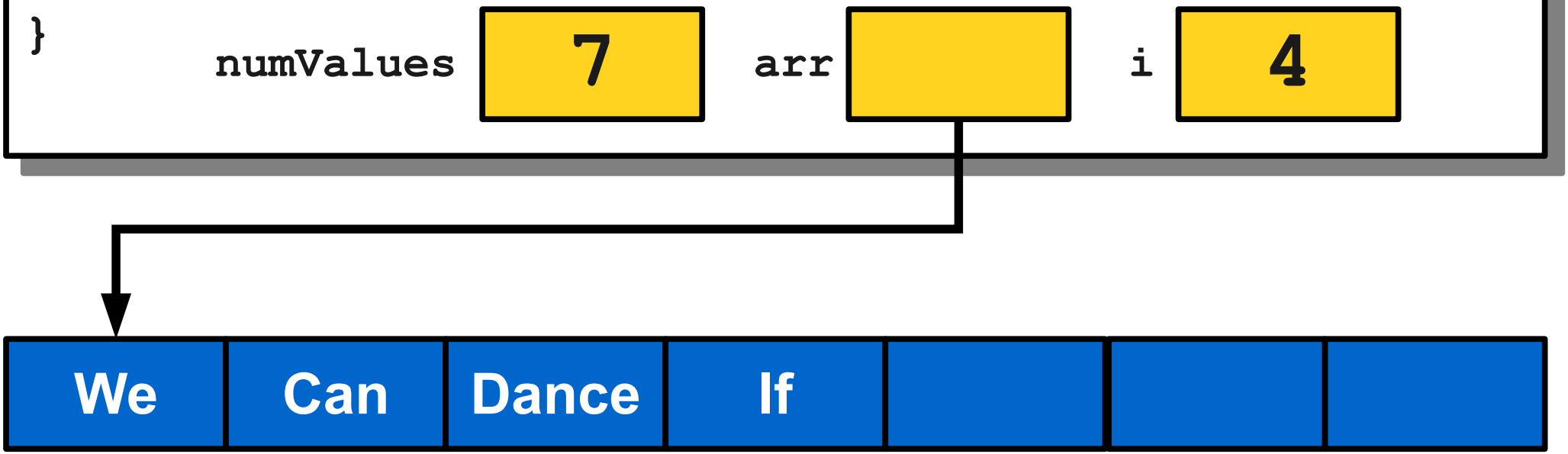
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



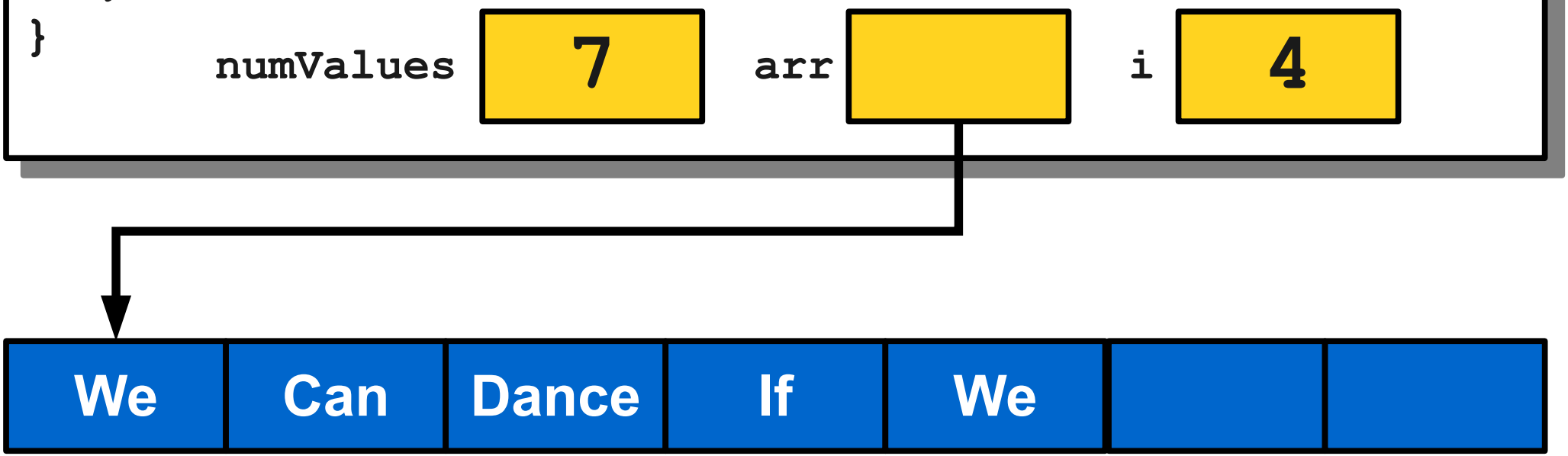
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



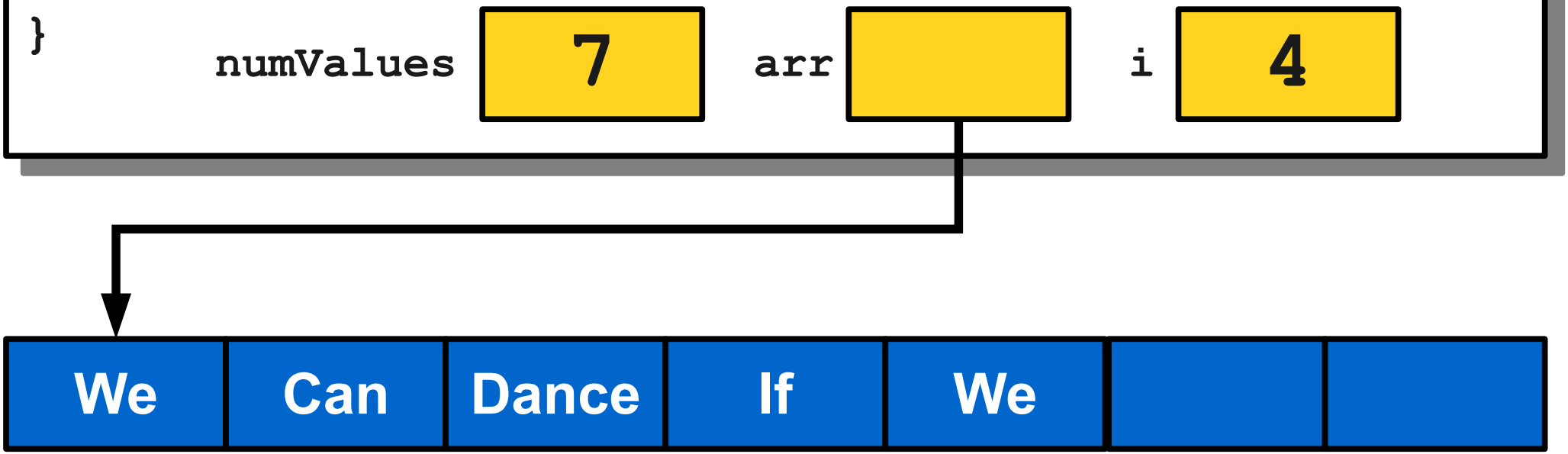
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



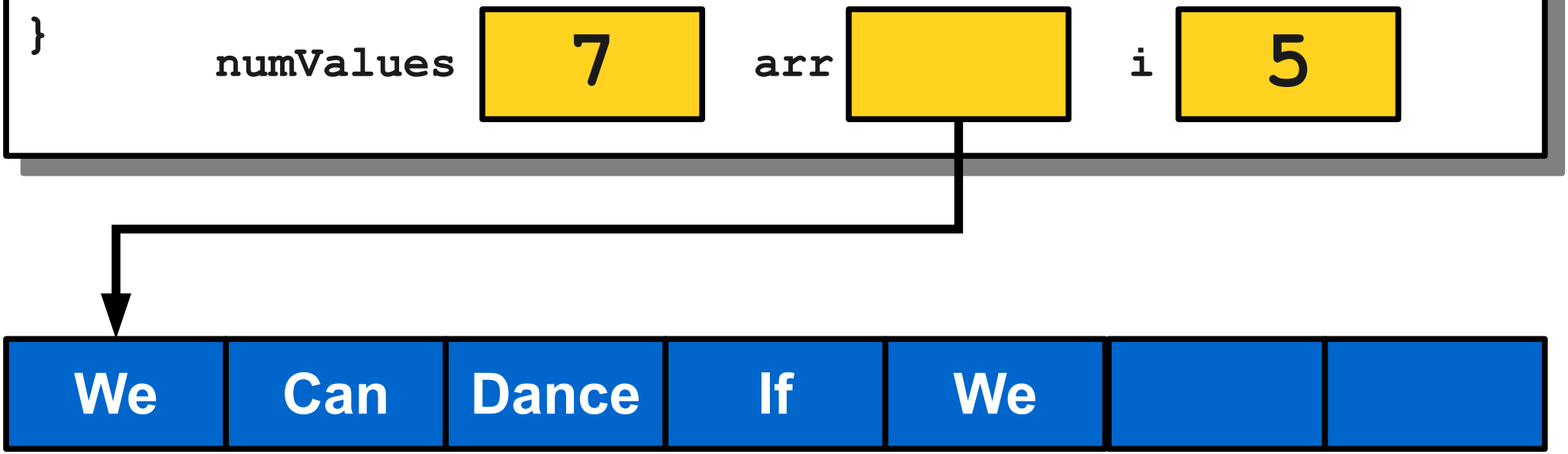
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



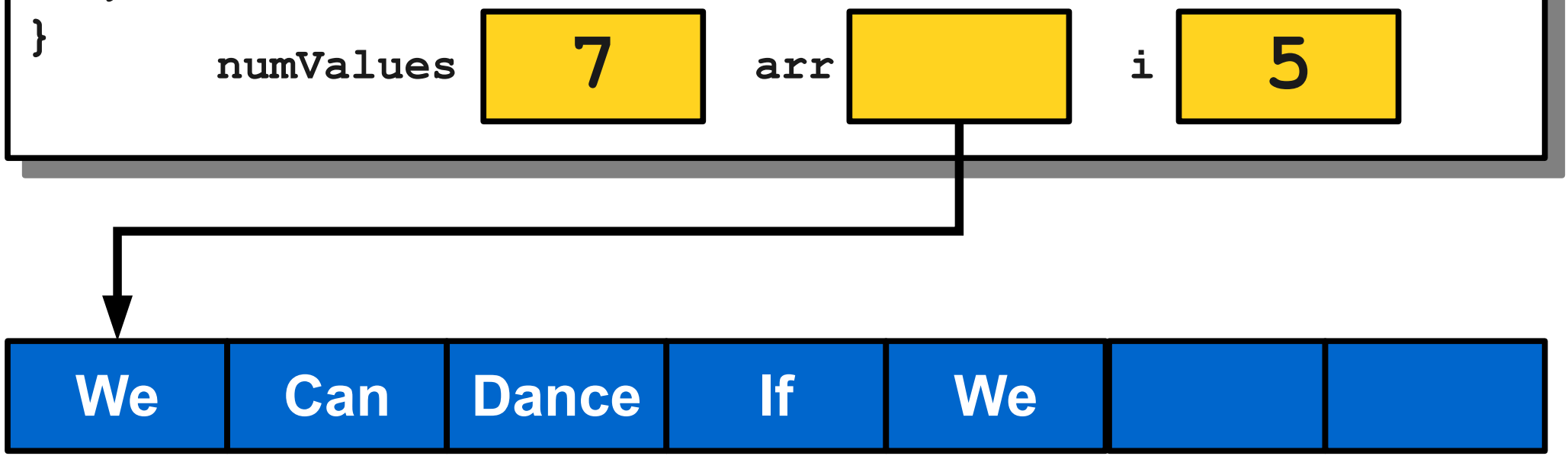
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



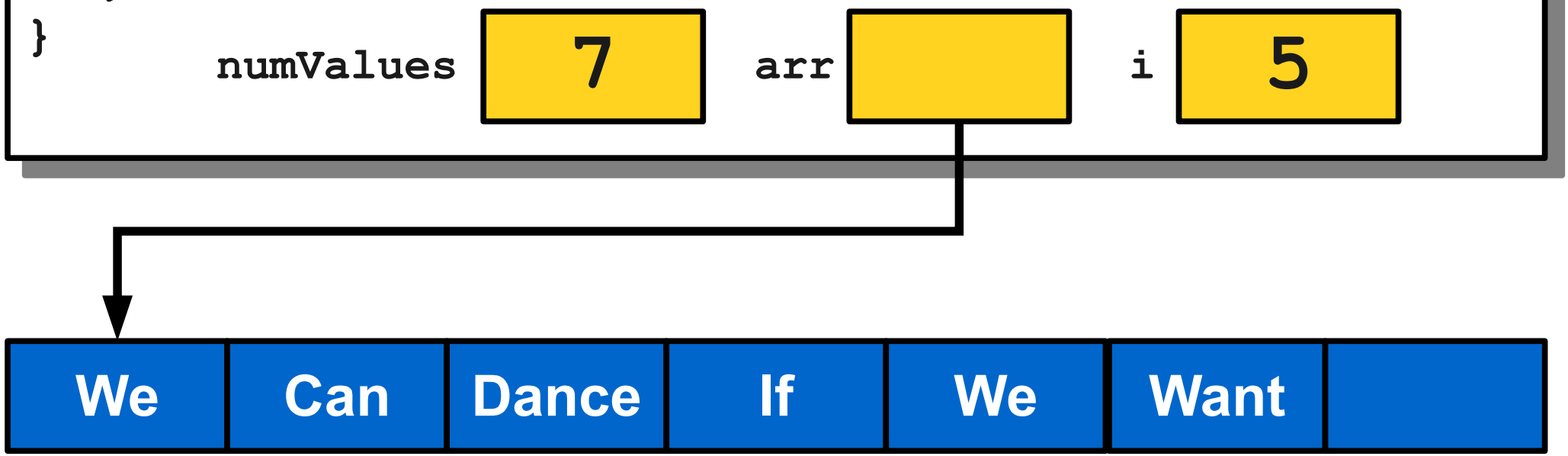
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



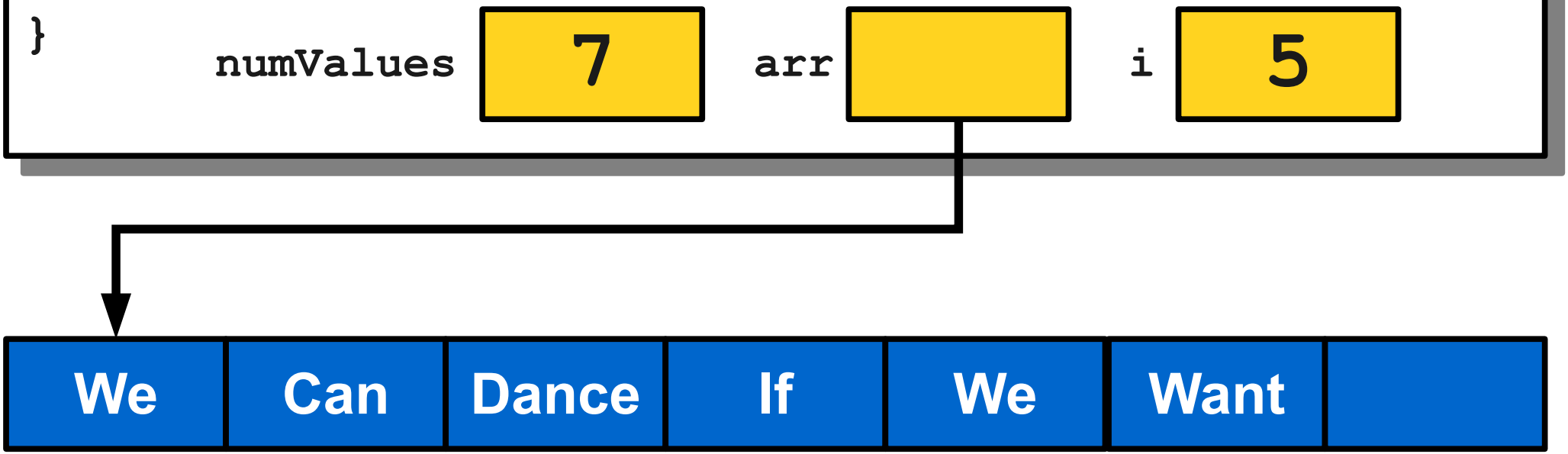
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



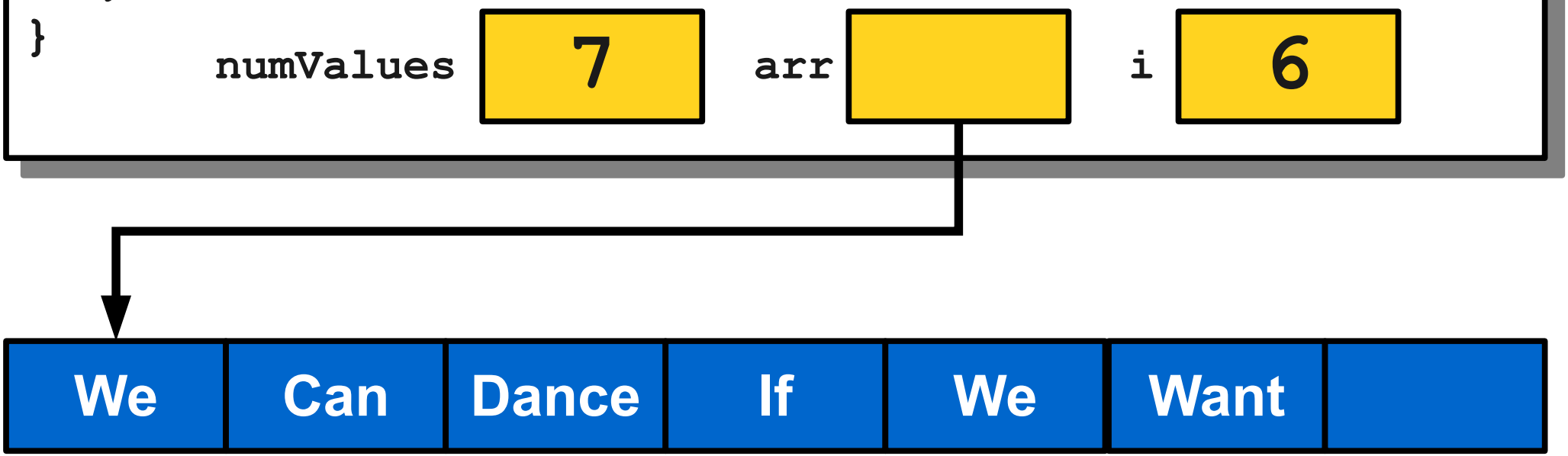
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



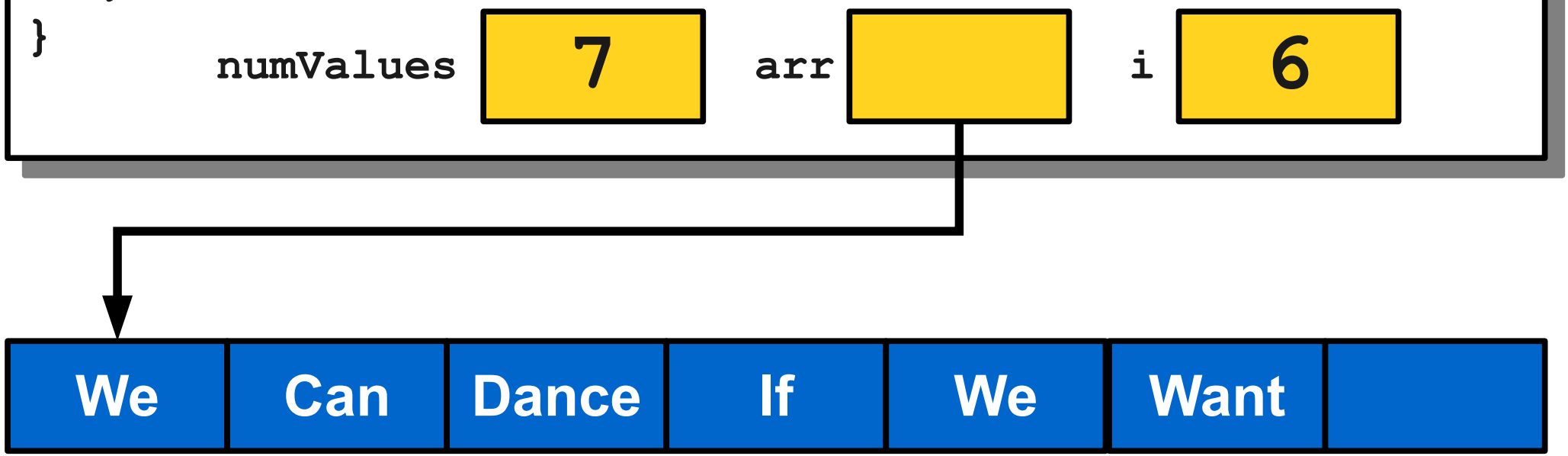
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



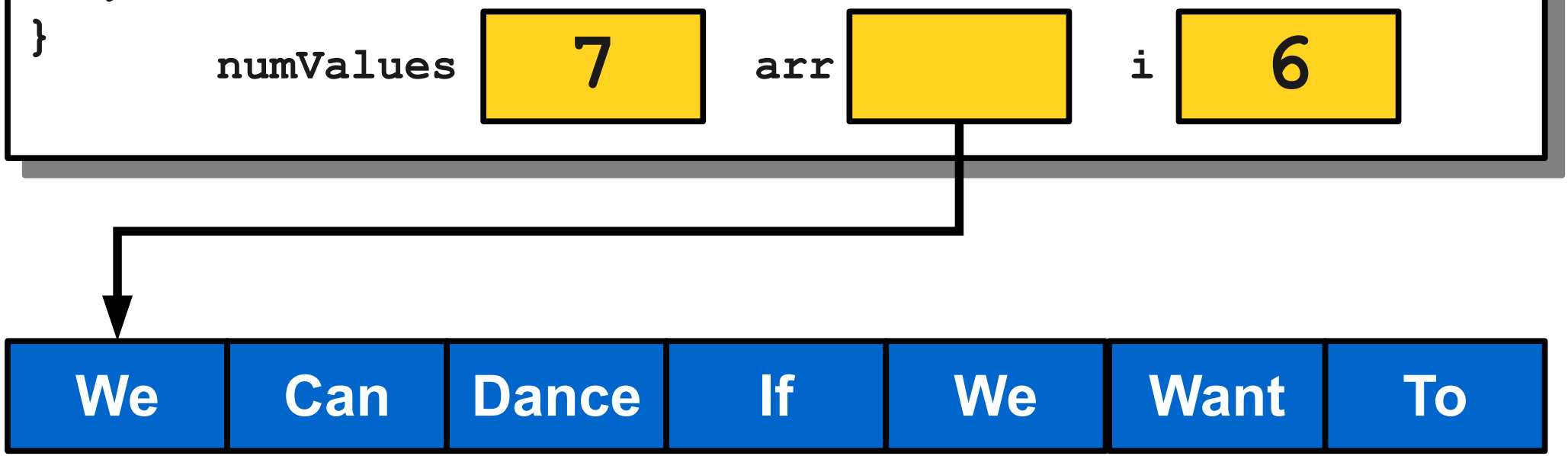
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



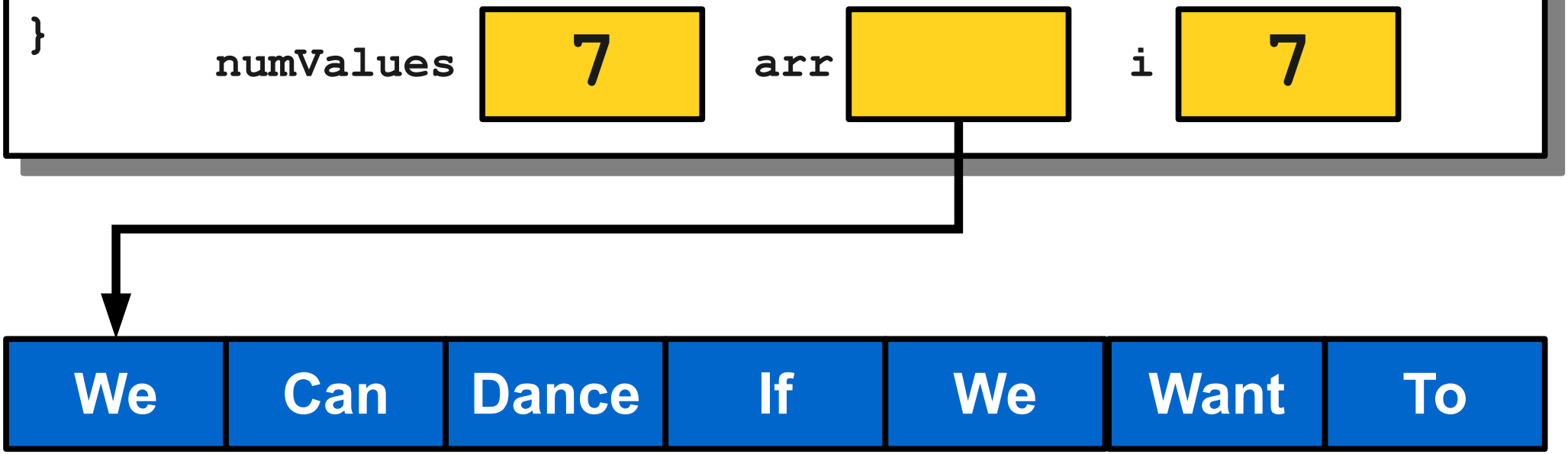
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



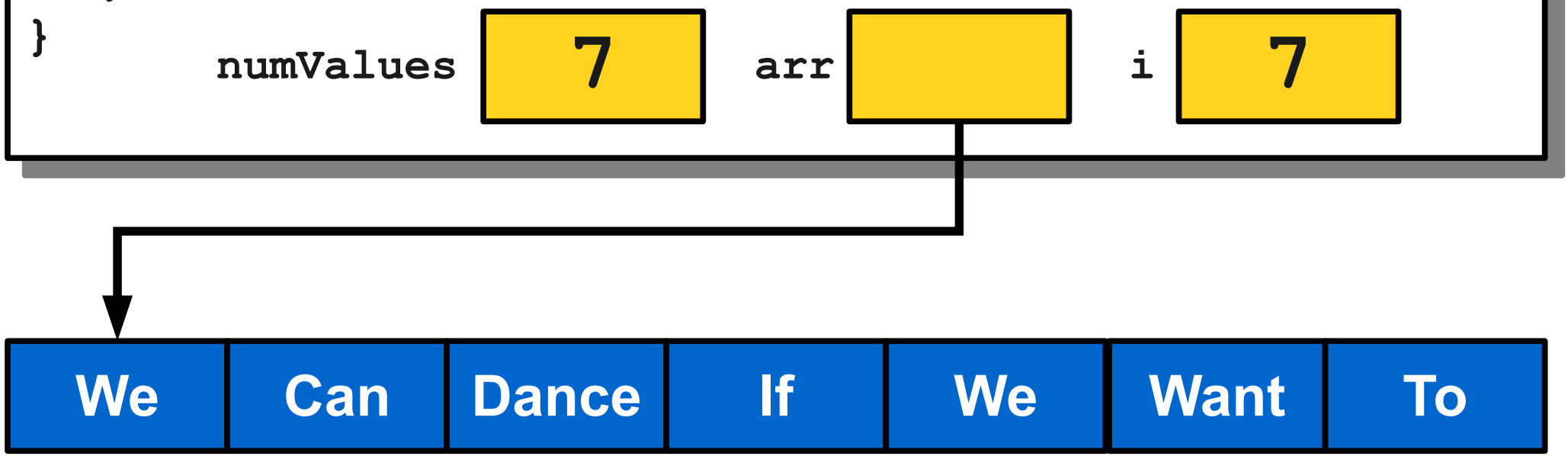
```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

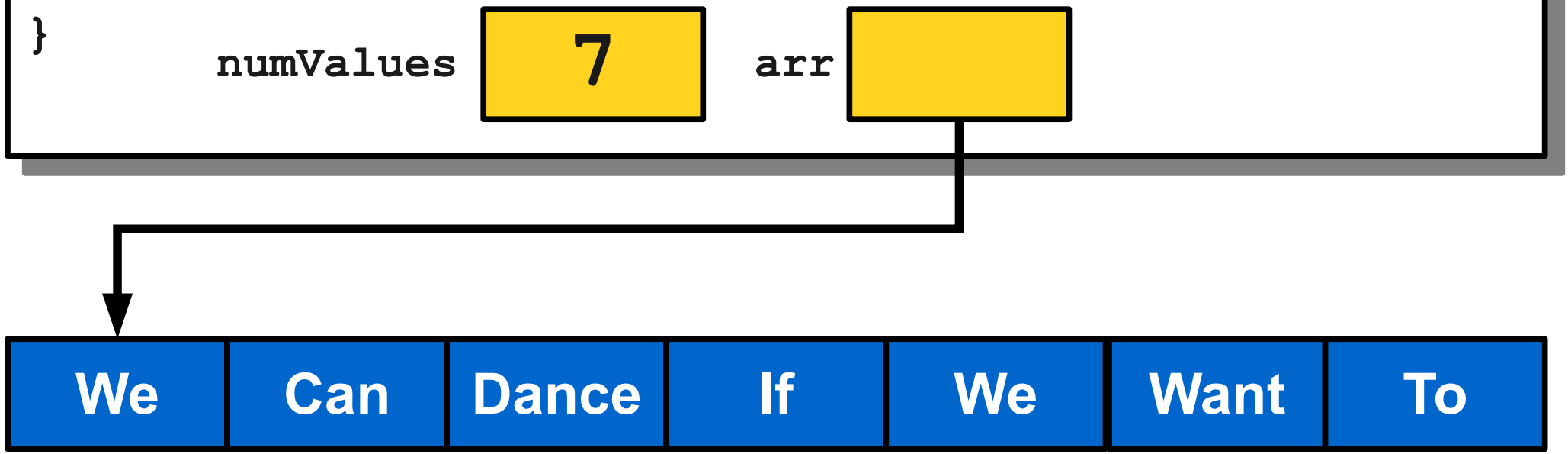


```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```

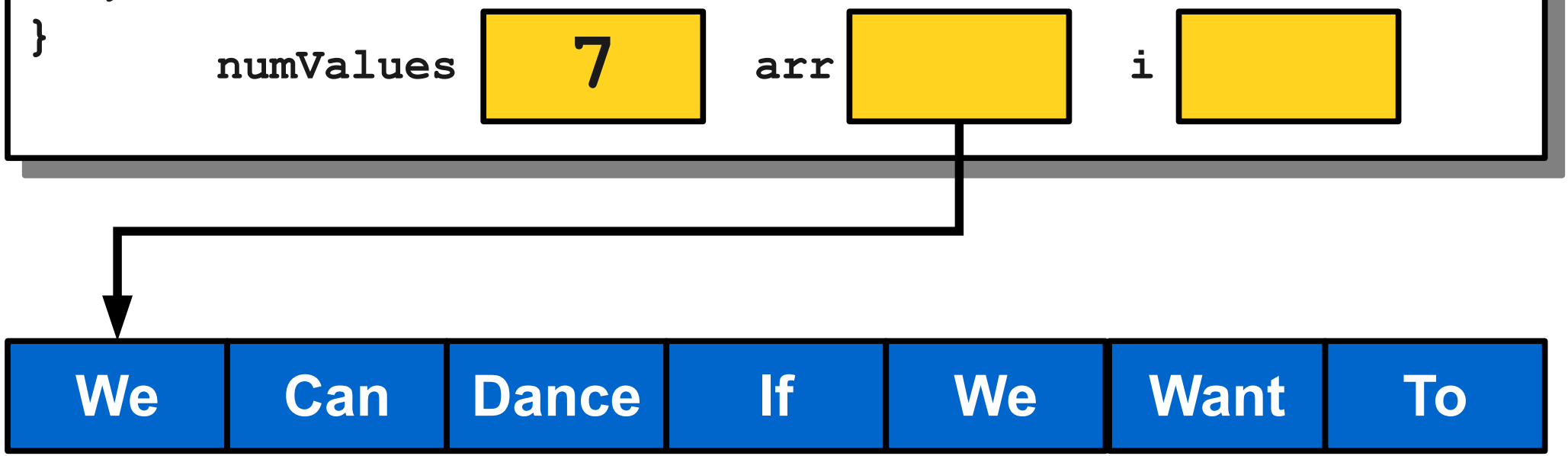
numValues 7

arr

```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



```
int main() {
   int numValues = getInteger("How many lines? ");

   string* arr = new string[numValues];
   for (int i = 0; i < numValues; i++) {
      arr[i] = getLine();
   }

   for (int i = 0; i < numValues; i++) {
      cout << i << ": " << arr[i] << endl;
   }
}
```



Dynamically Allocating Arrays

- First, declare a variable that will point at the newly-allocated array. If the array elements have type ***T***, the pointer will have type ***T****.
- Then, create a new array with the **new** keyword and assign the pointer to point to it.
- In two separate steps:

```
T* arr;
arr = new T[size];
```

- Or, in the same line:

```
T* arr = new T[size];
```

Cleaning Up

- When declaring global variables or local variables, C++ will automatically handle memory allocation and deallocation for you.
- When using `new`, you are responsible for deallocating the memory you allocate.
- If you don't, you get a **memory leak** and will slowly exhaust all of memory.
- Eventually, the program will crash when you ask for more memory with `new`, because the program thinks all of memory is in use.

Cleaning Up

- You can deallocate memory with the **`delete[]`** operator:

```
delete[] ptr;
```

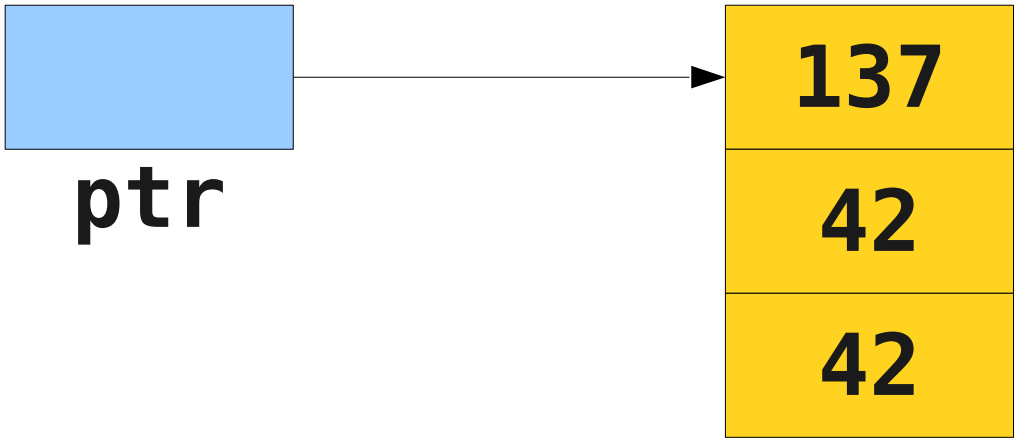
- This destroys the array pointed at by the given pointer, not the pointer itself.

Cleaning Up

- You can deallocate memory with the **`delete[]`** operator:

```
delete[] ptr;
```

- This destroys the array pointed at by the given pointer, not the pointer itself.

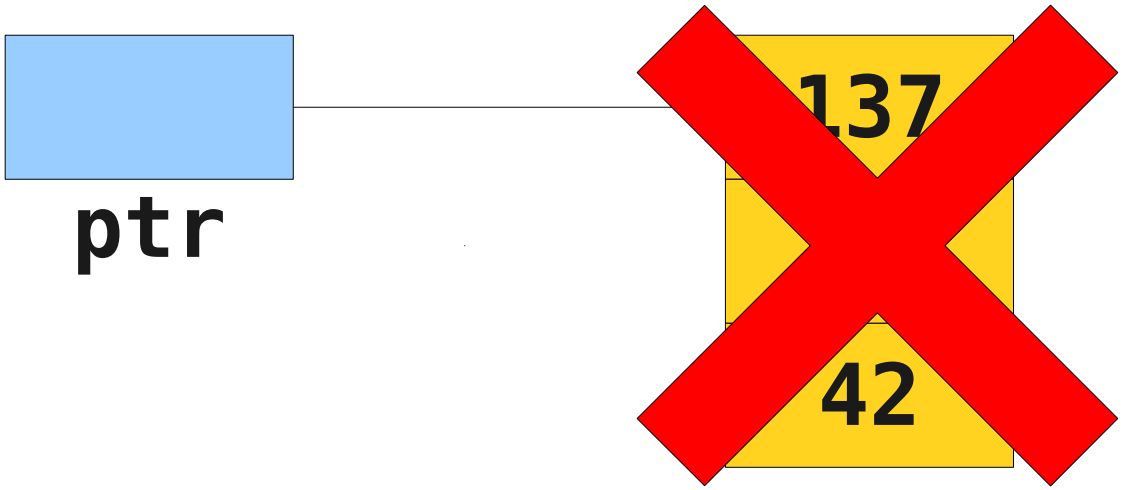


Cleaning Up

- You can deallocate memory with the **`delete[]`** operator:

```
delete[] ptr;
```

- This destroys the array pointed at by the given pointer, not the pointer itself.



Cleaning Up

- You can deallocate memory with the **`delete[]`** operator:

```
delete[] ptr;
```

- This destroys the array pointed at by the given pointer, not the pointer itself.

Words of Caution

- C++ has few of the safety features present in Java.
- All of the following result in **undefined behavior** in C++:
  - Reading or writing through a pointer that you haven't initialized.
  - Reading or writing through a pointer to an array that you have deallocated.
  - Reading or writing off the end of an array.

Implementing **`Stack`**

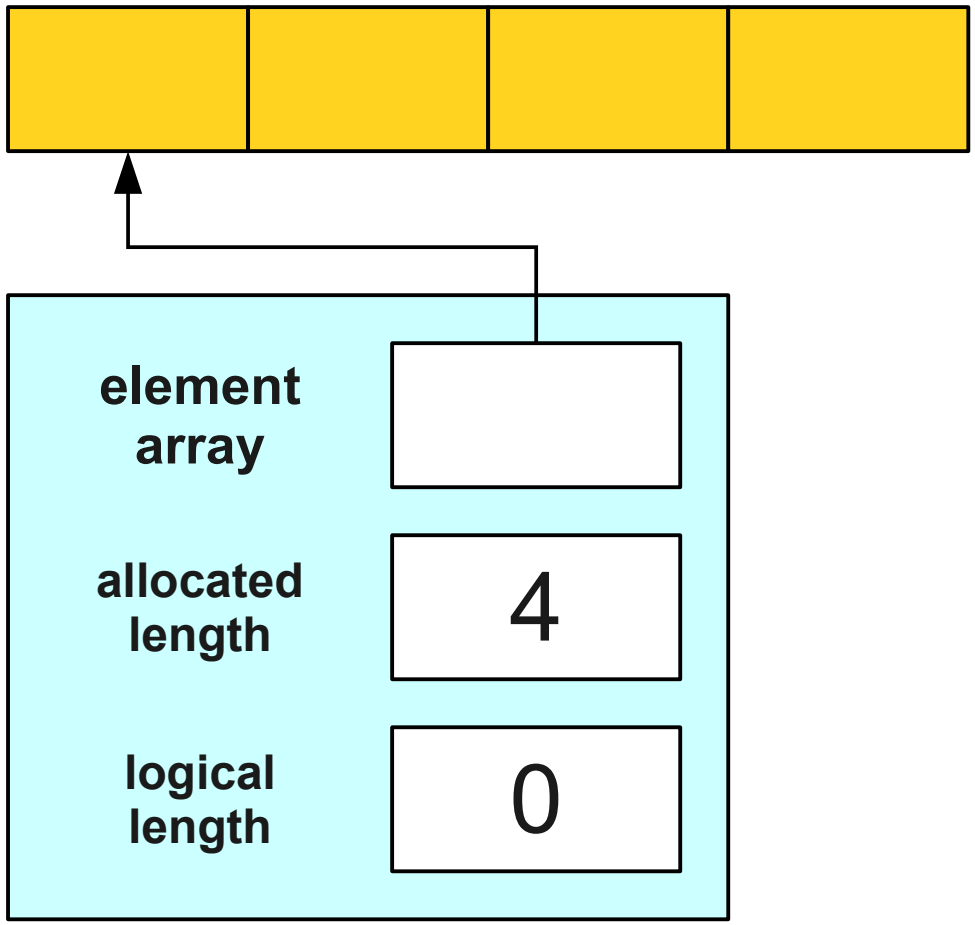
Implementing **`Stack`**

- Last time, we saw how to implement **`RandomBag`** in terms of **`Vector`**.
- We could also implement **`Stack`** in terms of **`Vector`**.
- What if we wanted to implement the **`Stack`** without relying on any other collections?
- Let's build the stack directly!

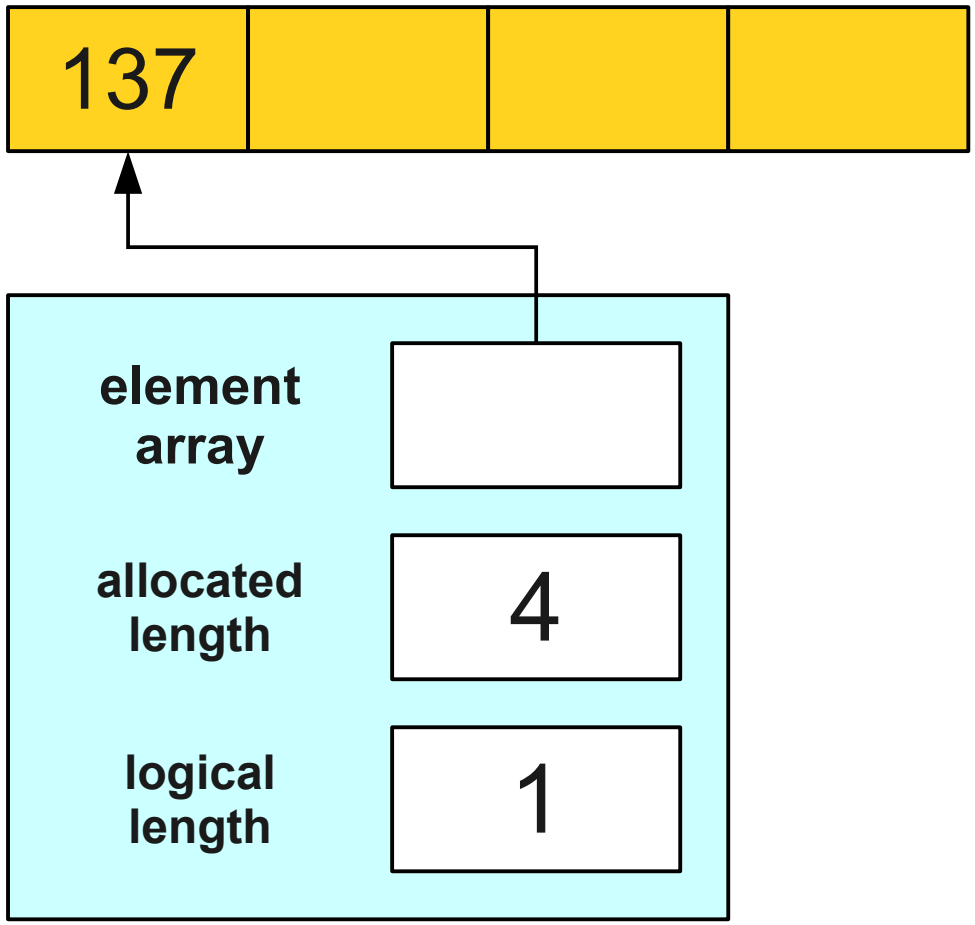
An Initial Idea

- **A bounded stack.**
- Allocate a fixed-size array for elements.
- Add elements to the array when they're pushed.
- Remove elements from the array when they're popped.
- Report an error if we exceed the size of the array.

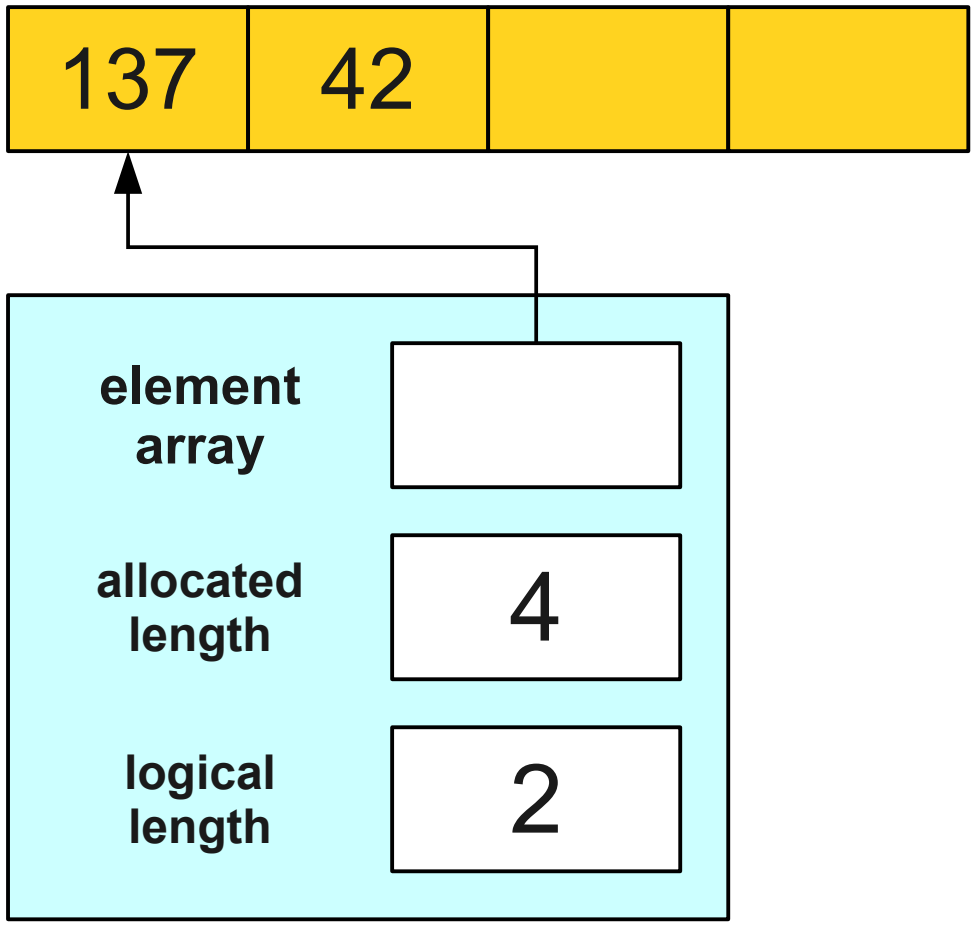
An Initial Idea



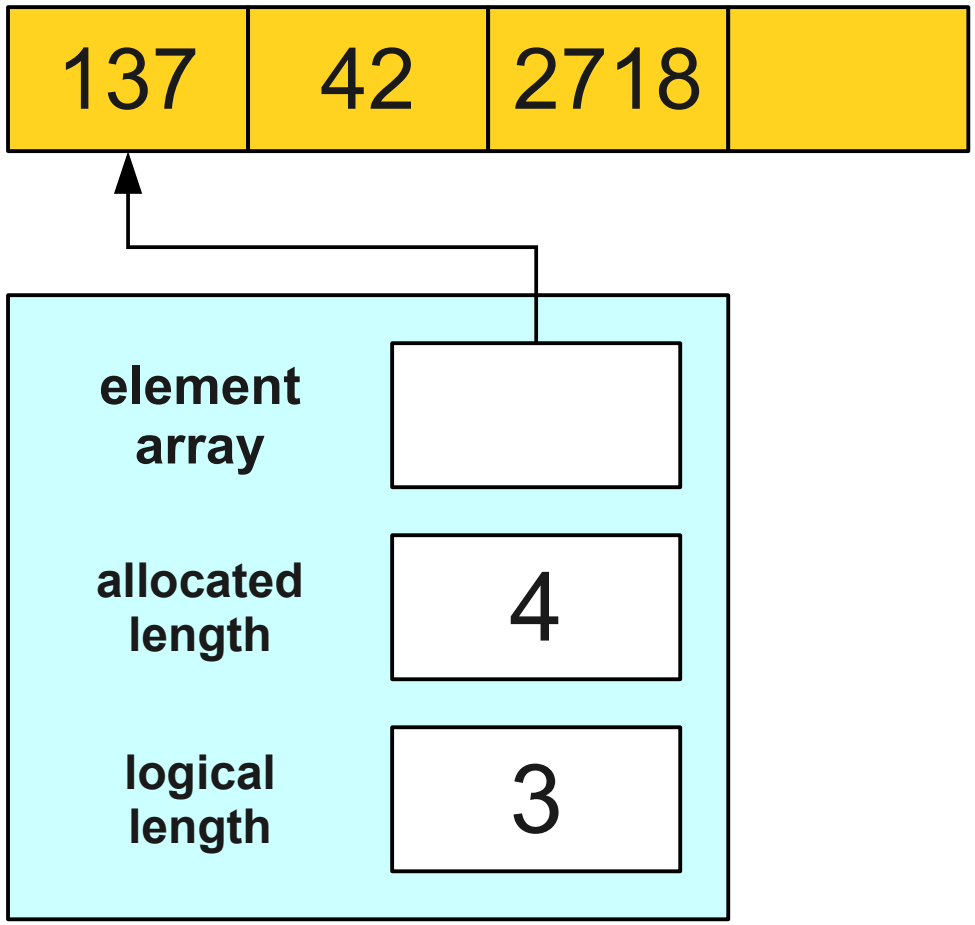
An Initial Idea



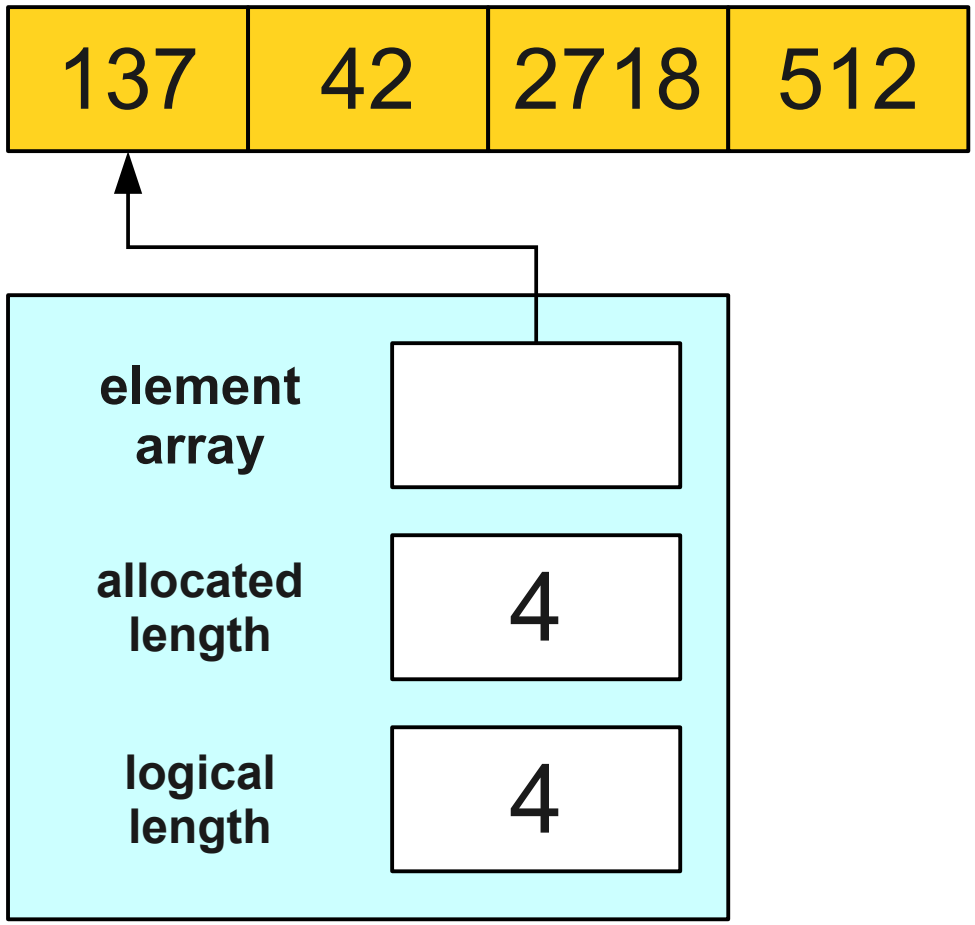
An Initial Idea



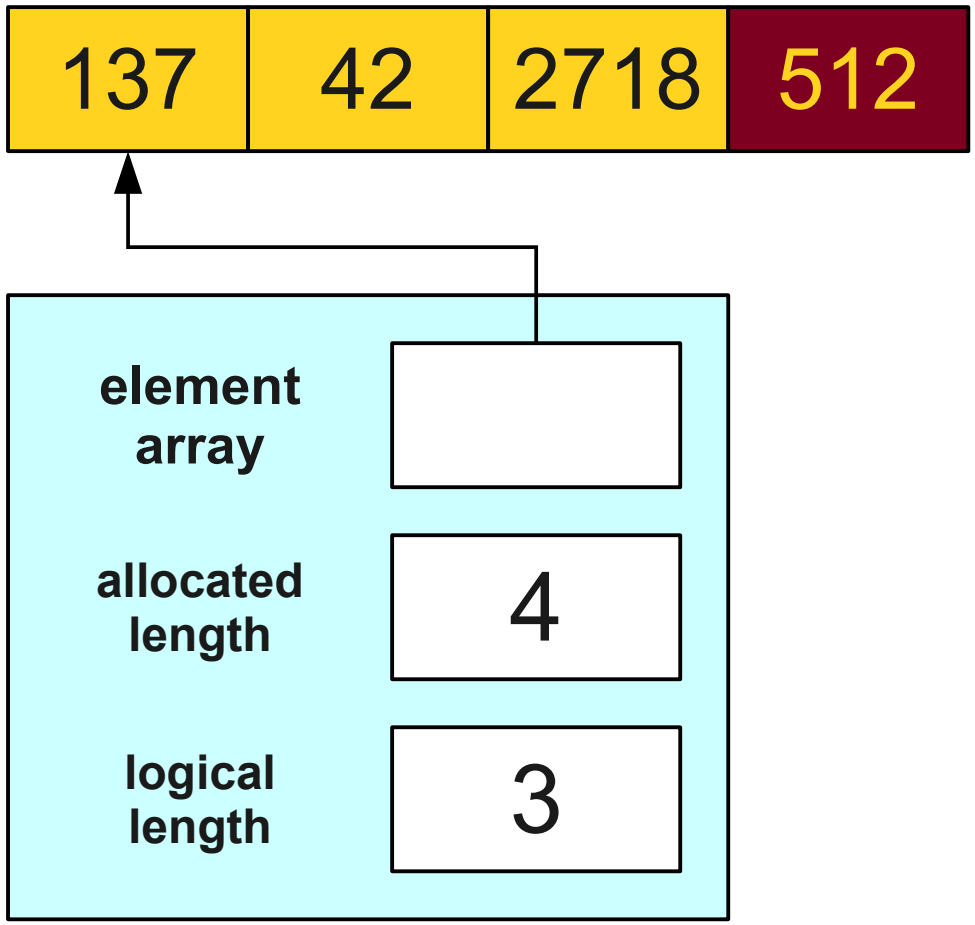
An Initial Idea



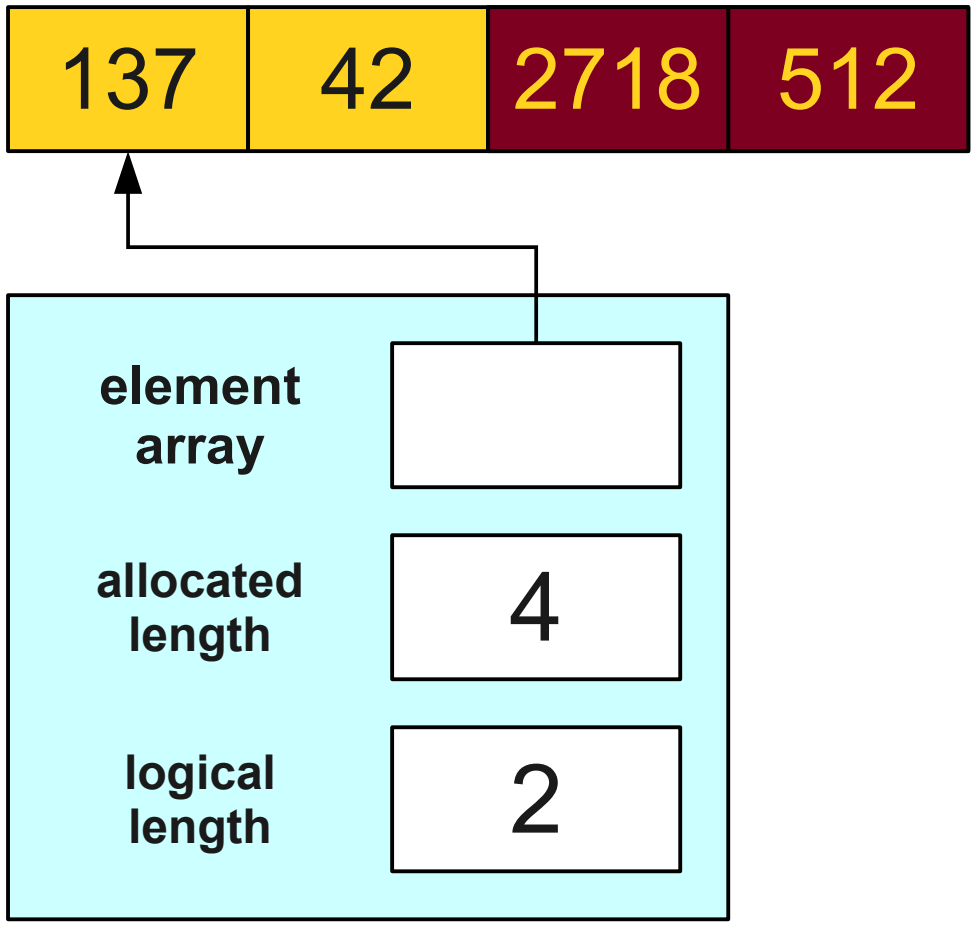
An Initial Idea



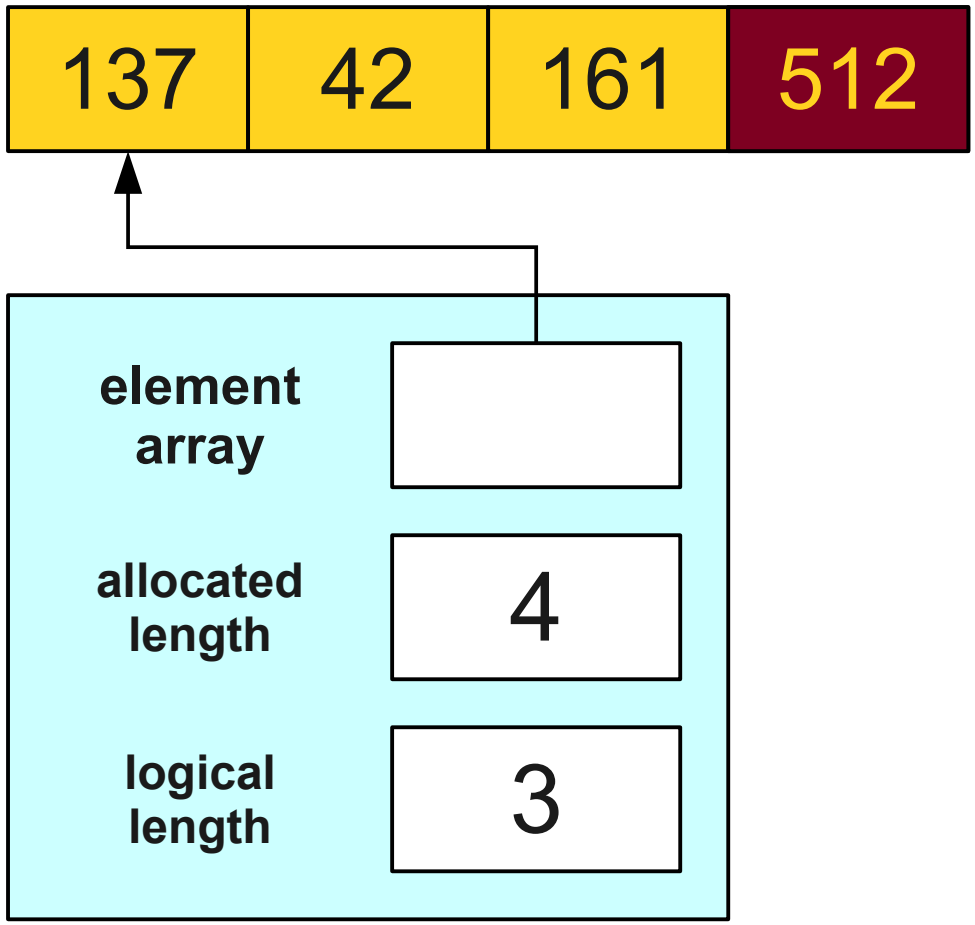
An Initial Idea



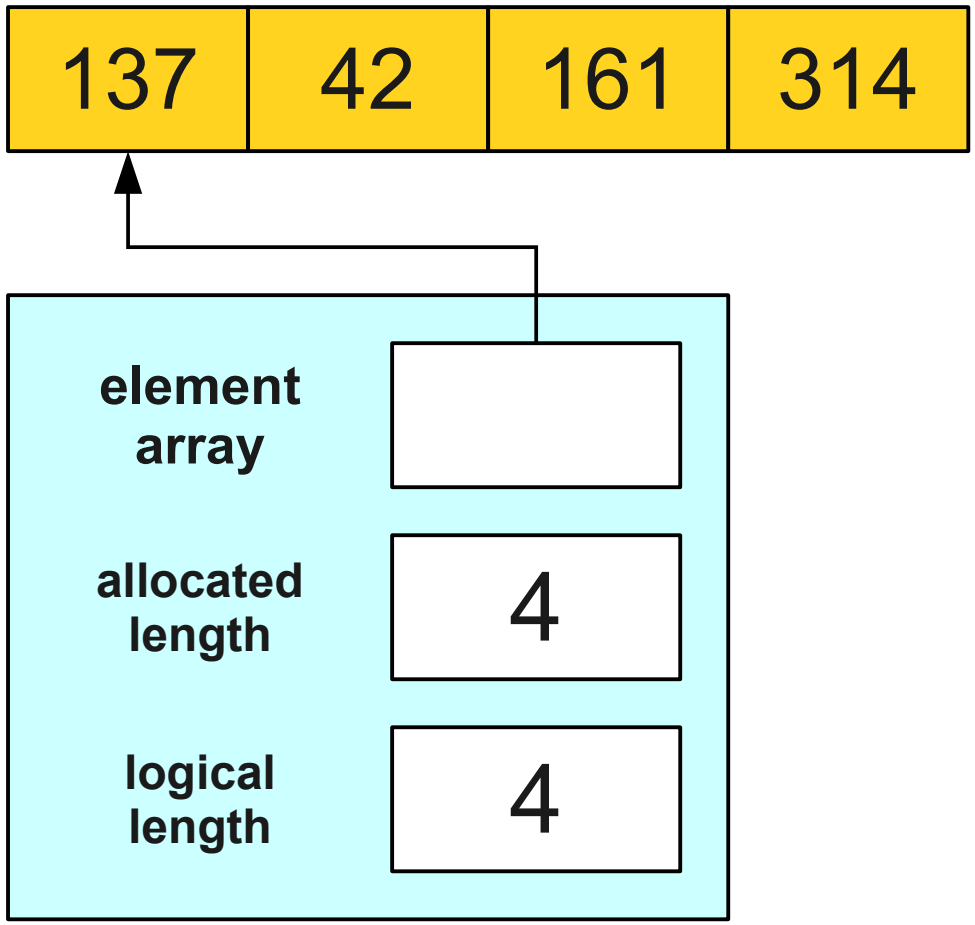
An Initial Idea



An Initial Idea



An Initial Idea



Let's Code it Up!

Constructors

- A **constructor** is a special member function used to set up the class before it is used.
- The constructor is automatically called when the object is created.
- Syntax: The constructor for a class named ***ClassName*** has signature

***ClassName*(*args*);**

Destructors

- A **destructor** is a special member function responsible for cleaning up an object's memory.
- Automatically called when a local variable goes out of scope.
- Automatically called if you `delete` a pointer to an object.
- Syntax: The destructor for a class named ***ClassName*** has signature

~*ClassName*();

Next Time

- **Making `Stack` Grow!**
  - Different approaches to **`Stack`** growth.
  - Analysis of these approaches.
  - The reality: *everything is a tradeoff!*