UPE Links

UPE Tutoring:

CS 31 Midterm 2 Review

Sign-in https://tinyurl.com/cs31mt2signin

Slides link available upon sign-in

Table of Contents

Pass by Value vs. Reference

Functions: Pass by Value

- By default, all parameters in C++ are pass by value.
- Every pass by value parameter is **copied** into the function

```
bool containsLowerCase(string s);

int main() {
  string s1 = “really long string”;
  containsLowerCase(s1);    // a copy of s1 is made and
}                           // passed to containsLowerCase
```

Functions: Pass by Reference

- A **reference** to a variable is passed to the function instead of a copy of the variable
- Syntax: add an & between parameter type and name
  - `int& x, bool& b, string& s`
- If these variables are **changed inside** the function, then they will also be **changed outside.**

Functions: swap

- Does this function properly swap the two variables passed to it?

```
void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}
```

Functions: swap

- Does this function properly swap the two variables passed to it?

```
void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}
```

No, it only swaps local copies! We need to use pass by reference.

Functions: swap #2

- Does this function properly swap the two variables passed to it?

```
void swap(int& x, int& y) {
  int temp = x;
  x = y;
  y = temp;
}
```

Functions: swap #2

- Does this function properly swap the two variables passed to it?

```
void swap(int& x, int& y) {
  int temp = x;
  x = y;
  y = temp;
}
```

Yes, because we are using the & modifier on the parameters to pass by reference.

Functions: Const Variables

- A parameter with the `const` modifier **cannot be modified** within the function. For example, we cannot change the value of `num` from within the body of the function `isPrime`.

```
bool isPrime(const int& num) {
  // Cannot change value of num here.
  ...
}
```

Functions: Const Variables

- Why are they useful?
  - Gives assurance to the caller of a function that the argument they pass in won't be modified
  - Makes convoluted functions easier to understand if we know a certain variable can't be modified

- These are usually **passed by reference.** *(It's a little weird to use it with pass by value.)*

Functions: Passing by Constant Reference

- The purpose of passing by reference is to save memory or allow modifications by the function.
- For cases where we want to avoid copying but don't want to allow functions to modify the variables we pass in.

Functions: Passing by Constant Reference

- If we pass by **const reference** we can:
  - avoid the cost of copying
  - prevent our variables from being modified by the function
- Essentially a free performance gain
- You'll run into const reference often in CS32

Practice Question: Pass By Reference

Assume that `foo` and `foo2` are implemented identically as follows:

```
int foo(int x, int& y, int& z,
    string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z <<
      endl;
    if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else {
    return 0;
}
```

Practice Question: Pass By Reference

What does this program print out?

```
int main() {
  int x = 1, y = 2, z = 3;
  string arr[] = {“Sup”, “Hi”, “Hey”};

  // Print values before the call to foo
  cout << x << “ “ << y << “ “ z << endl;
  // Print the return value of foo
  cout << foo(x, y, z, arr) << endl;
  // Print values after the call to foo
  cout << x << “ “ << y << “ “ << z << endl;
}
```

Walkthrough

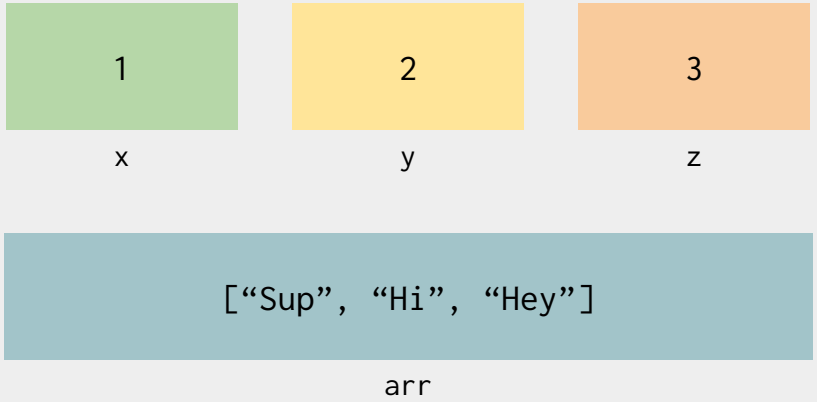
```
int main() {
  int x = 1, y = 2, z = 3;
  string arr[] = {“Sup”, “Hi”, “Hey”};

  // Print values before the call to foo
  cout << x << “ “ << y << “ “ z << endl;
  // Print the return value of foo
  cout << foo(x, y, z, arr) << endl;
  // Print values after the call to foo
  cout << x << “ “ << y << “ “ << z <<
      endl;
}
```

Walkthrough

```
int main() {
  int x = 1, y = 2, z = 3;
  string arr[] = {“Sup”, “Hi”, “Hey”};

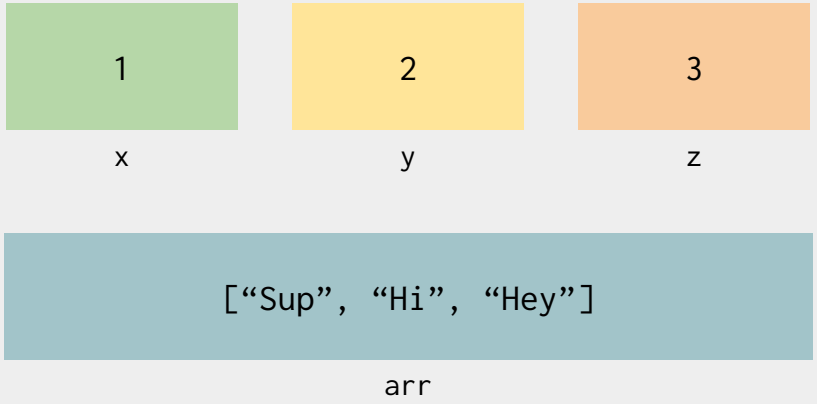
  // Print values before the call to foo
  cout << x << “ “ << y << “ “ z << endl;
  // Print the return value of foo
  cout << foo(x, y, z, arr) << endl;
  // Print values after the call to foo
  cout << x << “ “ << y << “ “ << z <<
      endl;
}
```



Walkthrough

```
int main() {
  int x = 1, y = 2, z = 3;
  string arr[] = {“Sup”, “Hi”, “Hey”};

  // Print values before the call to foo
  cout << x << “ “ << y << “ “ z << endl;
  // Print the return value of foo
  cout << foo(x, y, z, arr) << endl;
  // Print values after the call to foo
  cout << x << “ “ << y << “ “ << z <<
      endl;
}
```

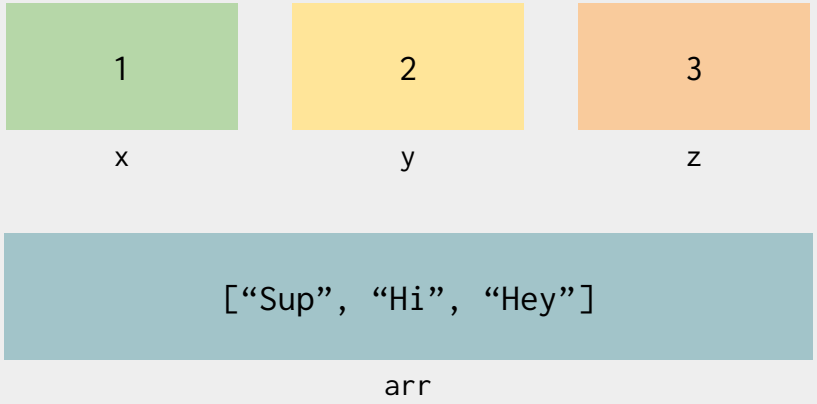



```
> 1 2 3
```

Walkthrough

```
int main() {
  int x = 1, y = 2, z = 3;
  string arr[] = {“Sup”, “Hi”, “Hey”};

  // Print values before the call to foo
  cout << x << “ “ << y << “ “ z << endl;
  // Print the return value of foo
  cout << foo(x, y, z, arr) << endl;
  // Print values after the call to foo
  cout << x << “ “ << y << “ “ << z <<
      endl;
}
```



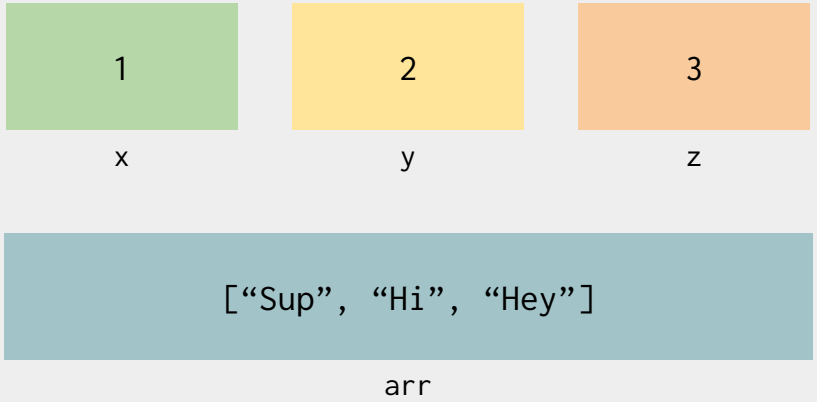

```
> 1 2 3
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



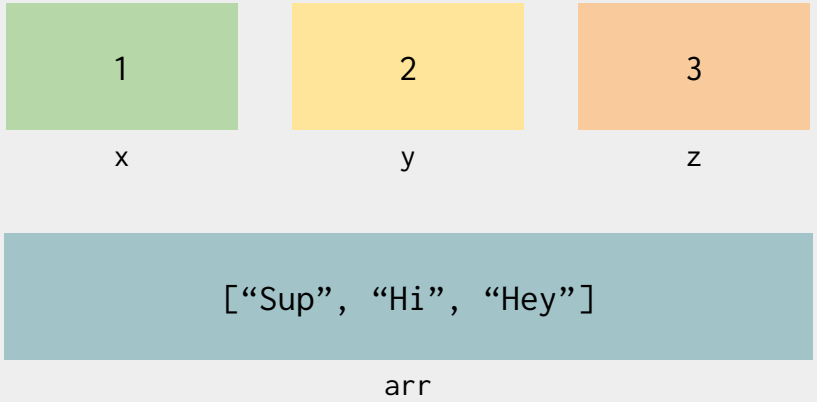

```
> 1 2 3
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



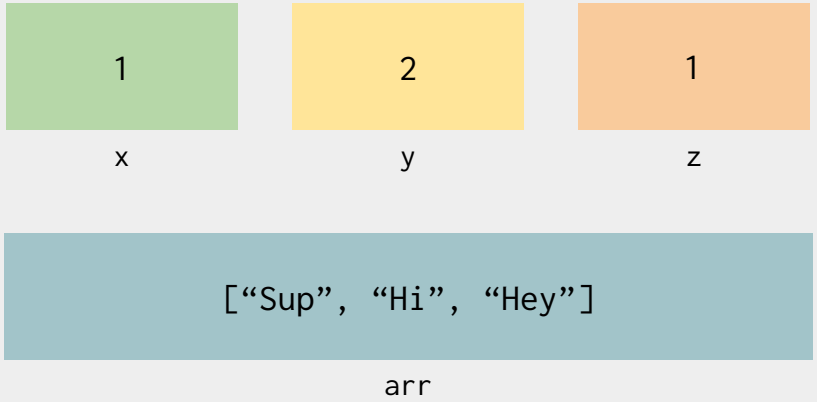

```
> 1 2 3
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



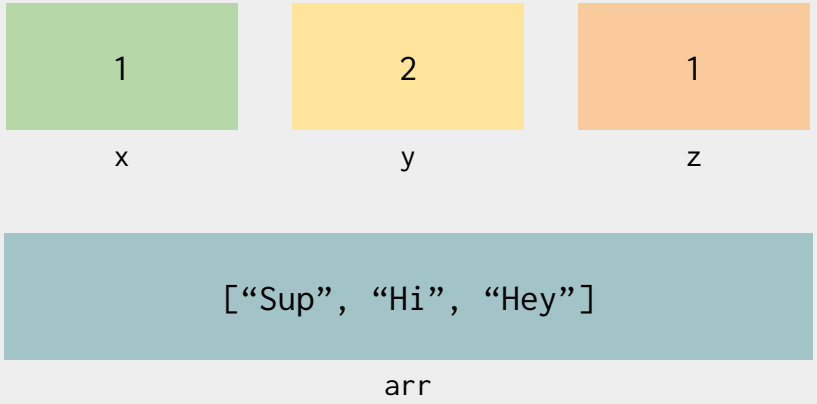

```
> 1 2 3
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



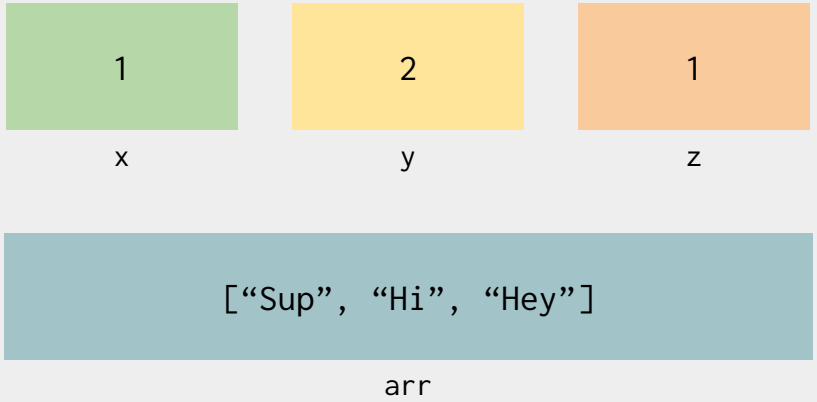

```
> 1 2 3
> 1 2 1
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



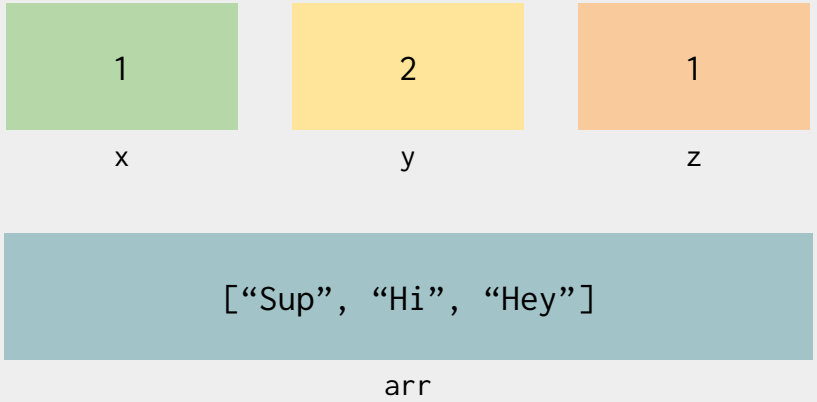

```
> 1 2 3
> 1 2 1
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



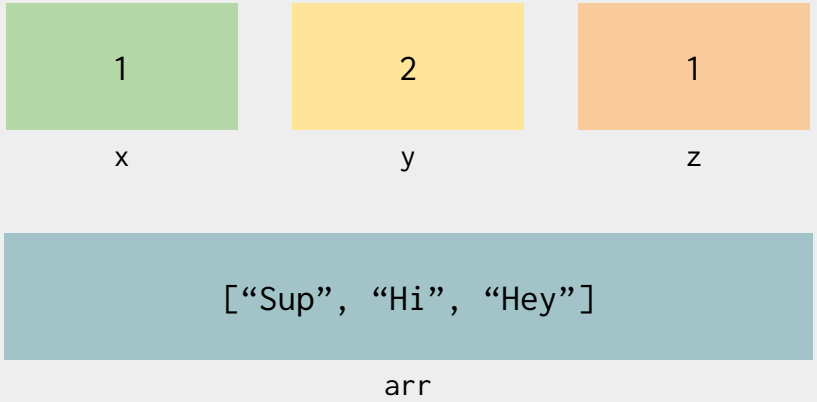

```
> 1 2 3
> 1 2 1
```

Walkthrough

```
int foo2(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



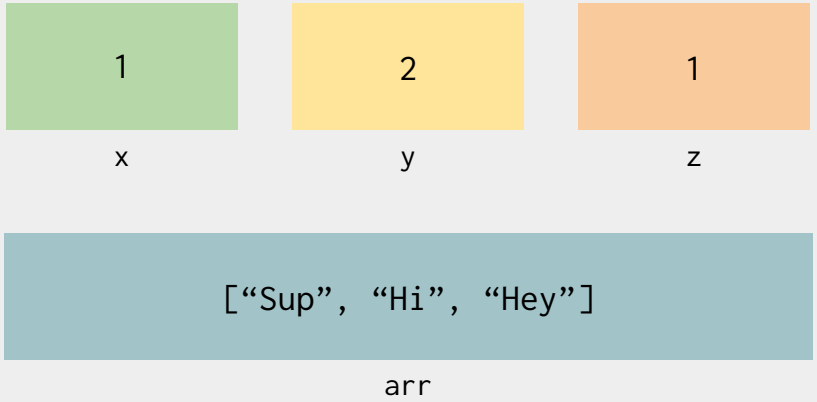

```
> 1 2 3
> 1 2 1
```

Walkthrough

```
int foo2(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



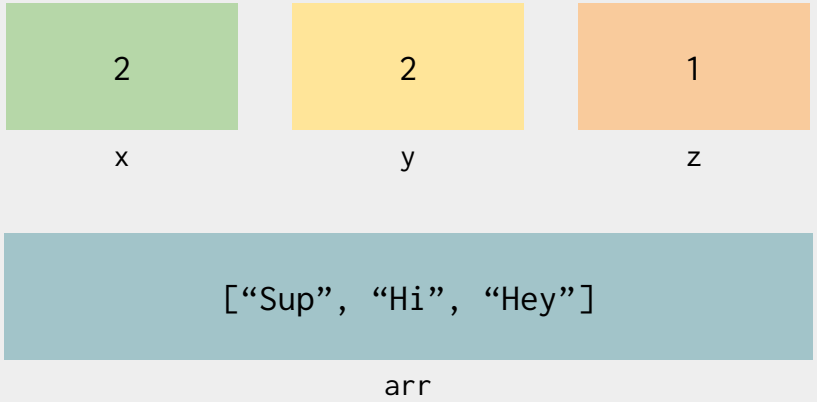

```
> 1 2 3
> 1 2 1
```

Walkthrough

```
int foo2(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



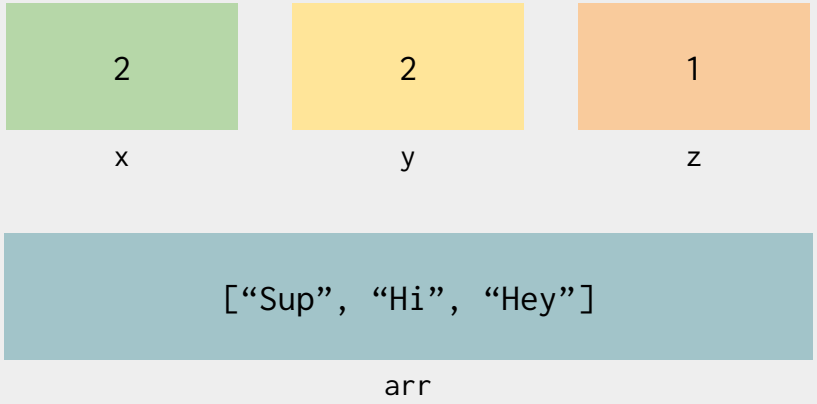

```
> 1 2 3
> 1 2 1
```

Walkthrough

```
int foo2(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



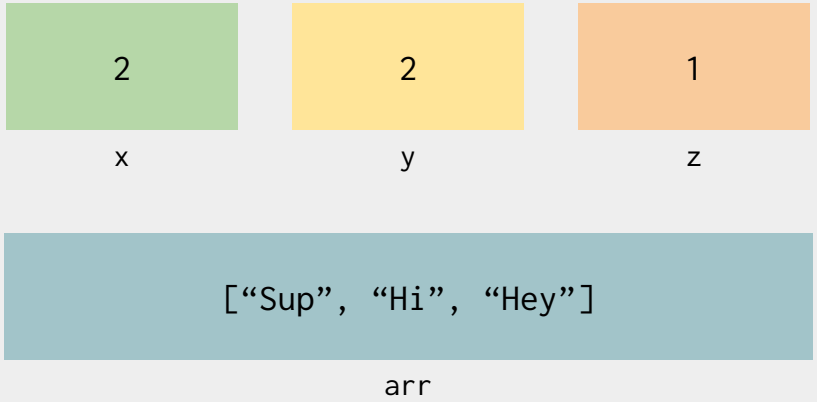

```
> 1 2 3
> 1 2 1
> 2 2 1
```

Walkthrough

```
int foo2(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



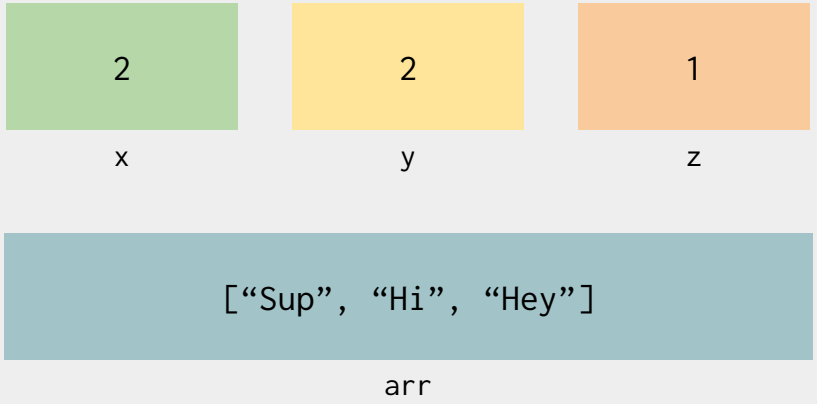

```
> 1 2 3
> 1 2 1
> 2 2 1
```

Walkthrough

```
int foo2(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



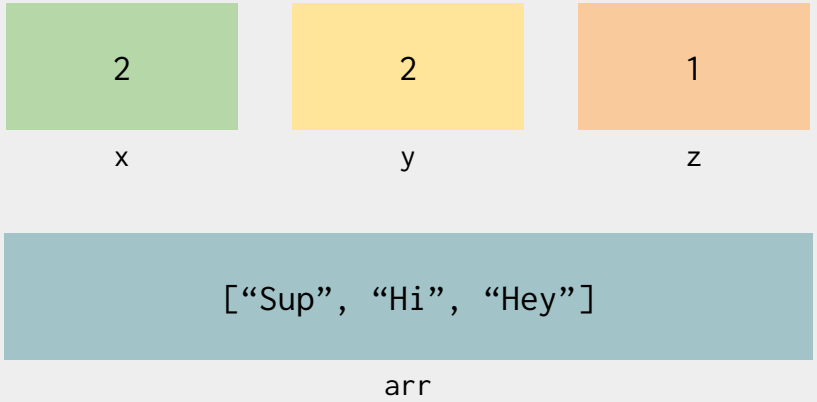

```
> 1 2 3
> 1 2 1
> 2 2 1
```

Walkthrough

```
int foo2(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



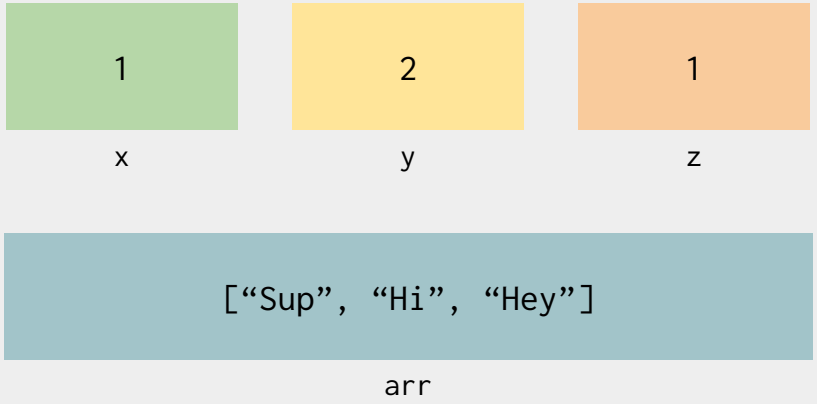

```
> 1 2 3
> 1 2 1
> 2 2 1
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```




```
> 1 2 3
> 1 2 1
> 2 2 1
```

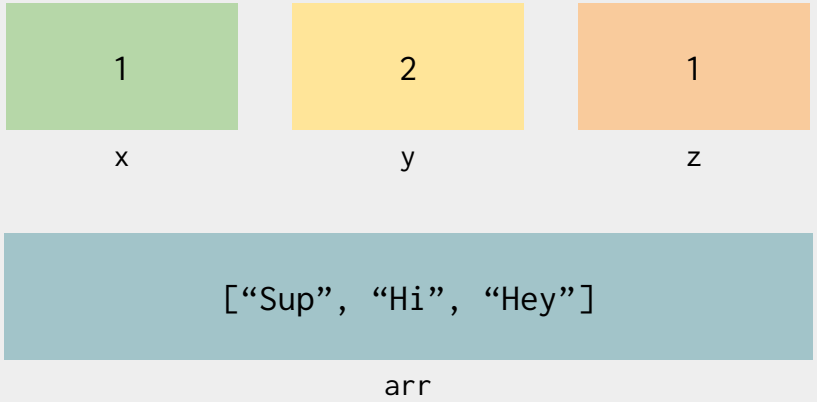
```
> 0
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```




```
> 1 2 3
> 1 2 1
> 2 2 1
```

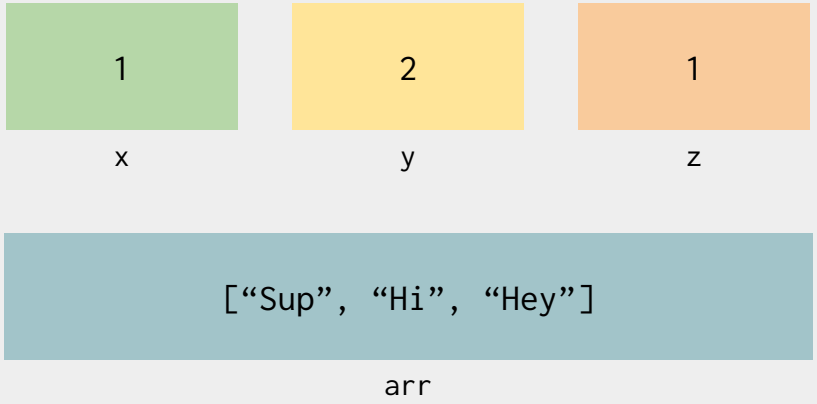
```
> 0
```

Walkthrough

```
int foo(int x, int& y, int& z, string arr[]) {
  if (x == z)
    x = y;
  else
    z = x;
  cout << x << " " << y << " " << z << endl;

  if (x != y)
    cout << foo2(x, y, z, arr) << endl;

  if (arr[x-1] == "Sup")
    return 1;
  else
    return 0;
}
```



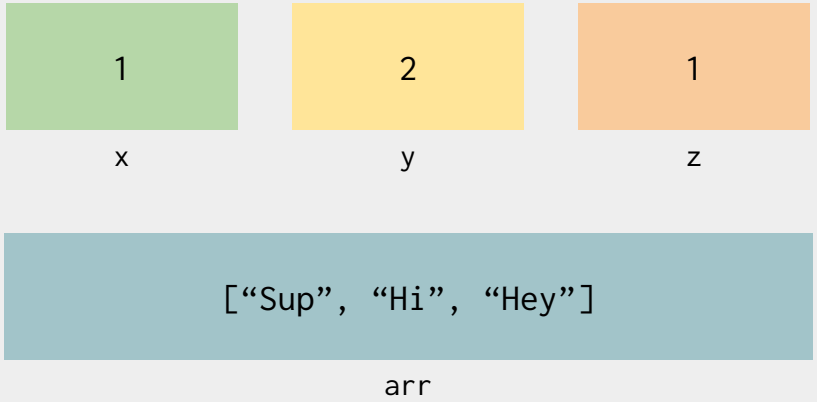

```
> 1 2 3
> 1 2 1
> 2 2 1
```

```
> 0
```

Walkthrough

```
int main() {
  int x = 1, y = 2, z = 3;
  string arr[] = {“Sup”, “Hi”, “Hey”};

  // Print values before the call to foo
  cout << x << “ “ << y << “ “ z << endl;
  // Print the return value of foo
  cout << foo(x, y, z, arr) << endl;
  // Print values after the call to foo
  cout << x << “ “ << y << “ “ << z <<
      endl;
}
```



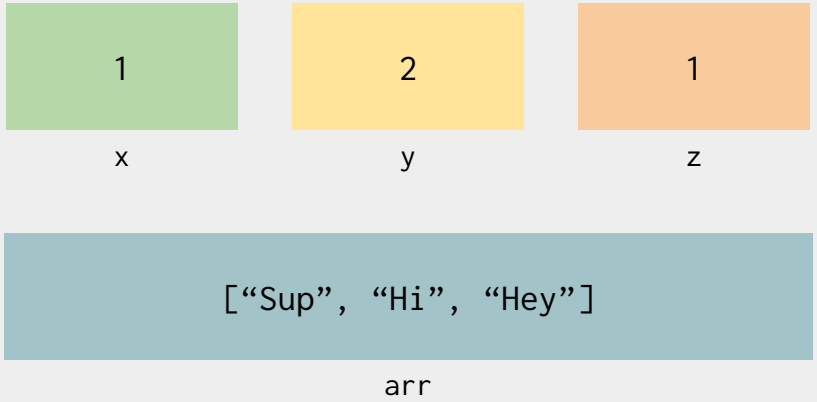

```
> 1 2 3
> 1 2 1
> 2 2 1
```

```
> 0
> 1
```

Walkthrough

```
int main() {
  int x = 1, y = 2, z = 3;
  string arr[] = {“Sup”, “Hi”, “Hey”};

  // Print values before the call to foo
  cout << x << “ “ << y << “ “ z << endl;
  // Print the return value of foo
  cout << foo(x, y, z, arr) << endl;
  // Print values after the call to foo
  cout << x << “ “ << y << “ “ << z <<
      endl;
}
```




```
> 1 2 3
> 1 2 1
> 2 2 1
```

```
> 0
> 1
> 1 2 1
```

Arrays

Arrays

- An array is a series of elements of the same type placed in contiguous memory locations
  - Elements can be accessed through their index
    - e.g. `nums[2]` (from below) is 3
  - Indexing of arrays starts from 0
- Valid declarations:

```
int arr[10];

bool list[5];

const int MAX_SIZE = 10;
string words[MAX_SIZE];

int nums[] = {1, 2, 3};

int arr[5] = {};  // initializes to all 0s
int arr[5] = {1, 2, 3};  // initializes to {1, 2, 3, 0, 0}
```

Arrays (cont.)

- Rules for specifying size:
  - **Must** be included in the brackets
  - **Cannot** involve a variable unless it is a constant known at compile time
  - The only time size can be left out is when a list of its contents is included
- Not allowed in C++:
  - `int arr[]; // Size not included.`
  - ```
    /****** Use of non-const variable. ******/
    int x;
    cin >> x;
    char buffer[x];
    ```
    ```

Passing Arrays to Functions

- Parameter Syntax
  - `(..., type name[], ...)`
- Arrays are default passed by reference
  - Any changes made to the array will be retained outside of the function scope

Passing Arrays to Functions (cont.)

- Size of array should be passed to the function
- Call to the function just passes in array name

```
// arr is the array itself, n is the size.
int firstOdd(int arr[], int n) {
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1)
      return i;
  }
  return n; // If no odd number found.
}
```

What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```

What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```

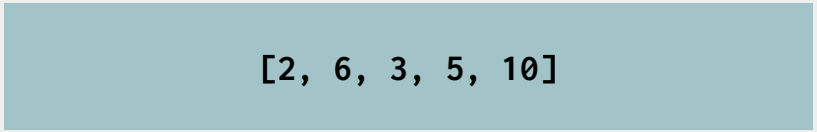
5

n

What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```

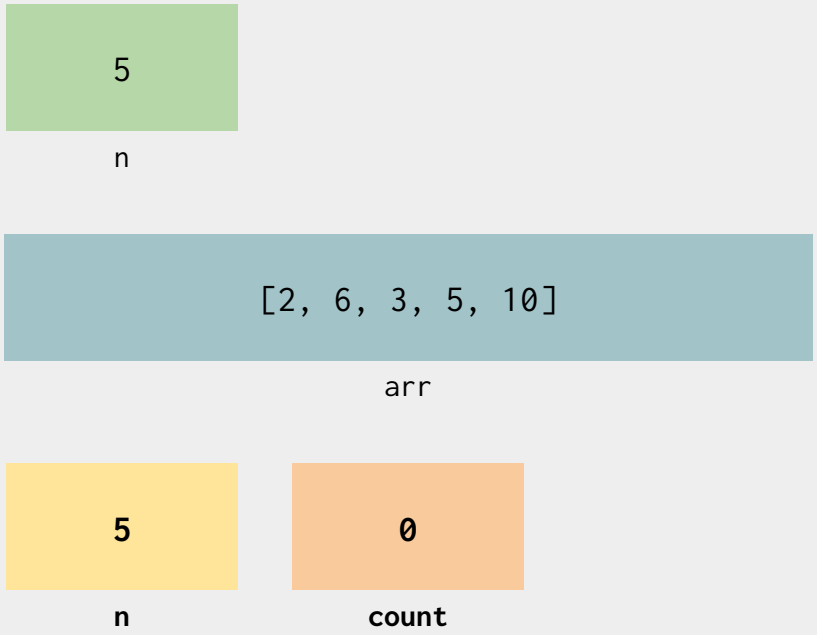
n



arr

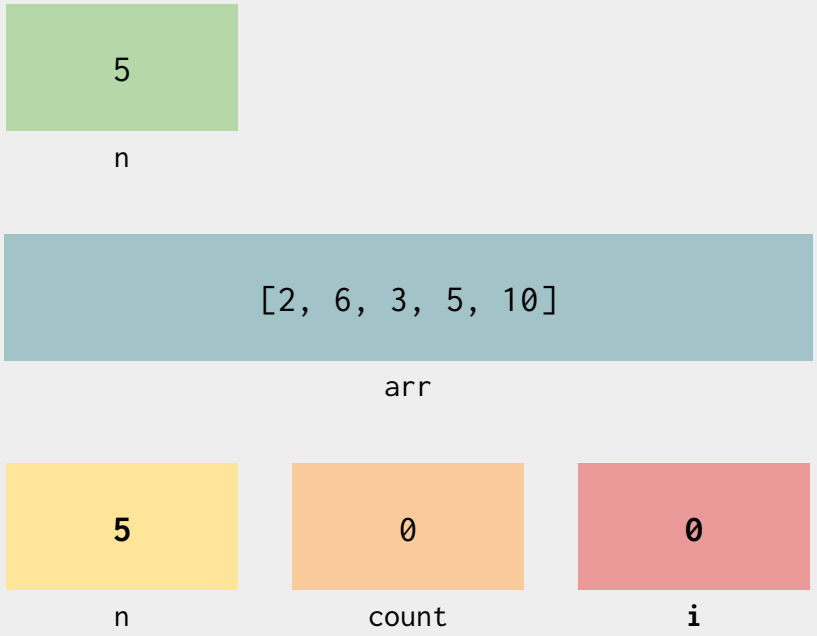
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



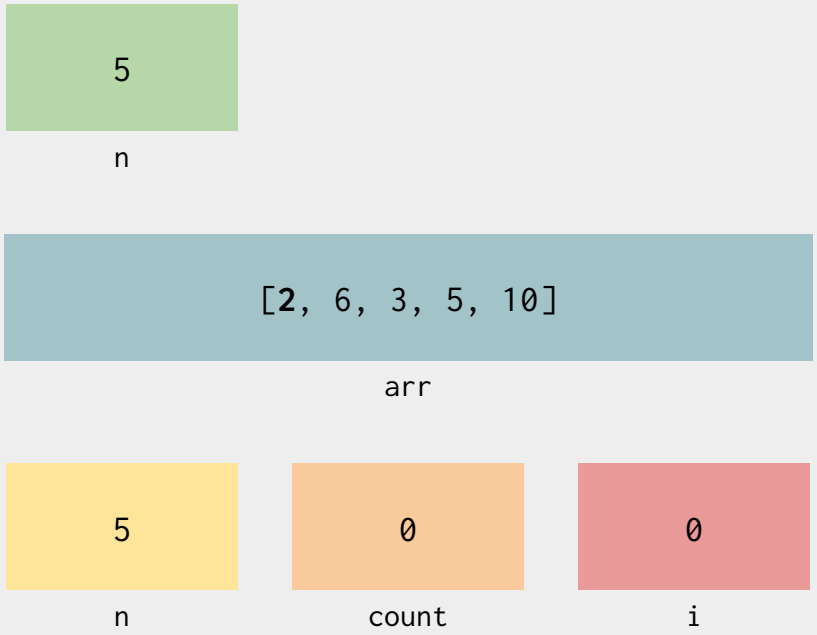
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



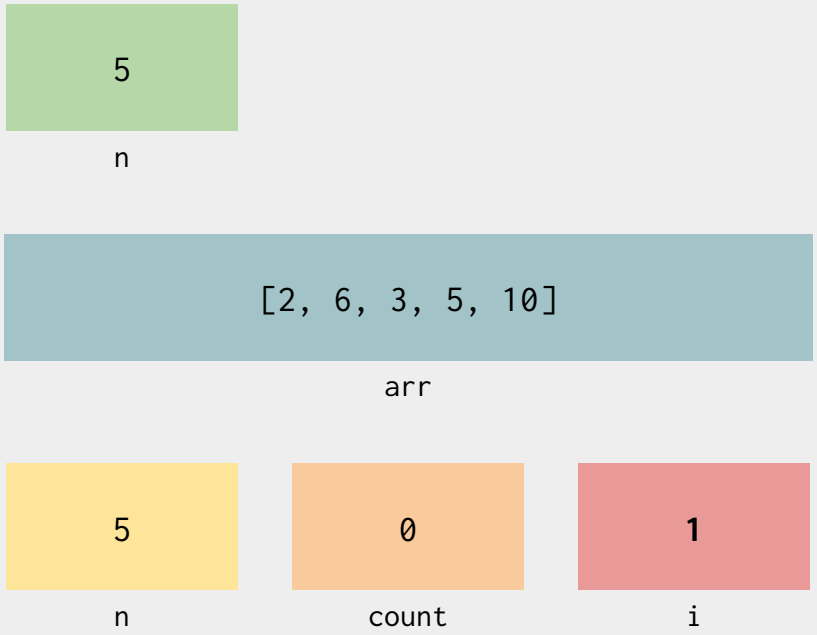
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



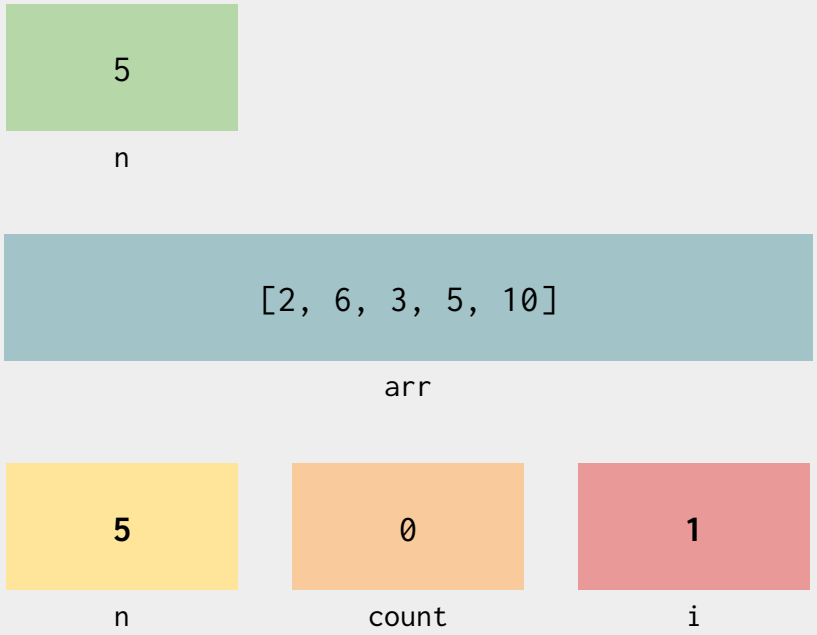
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



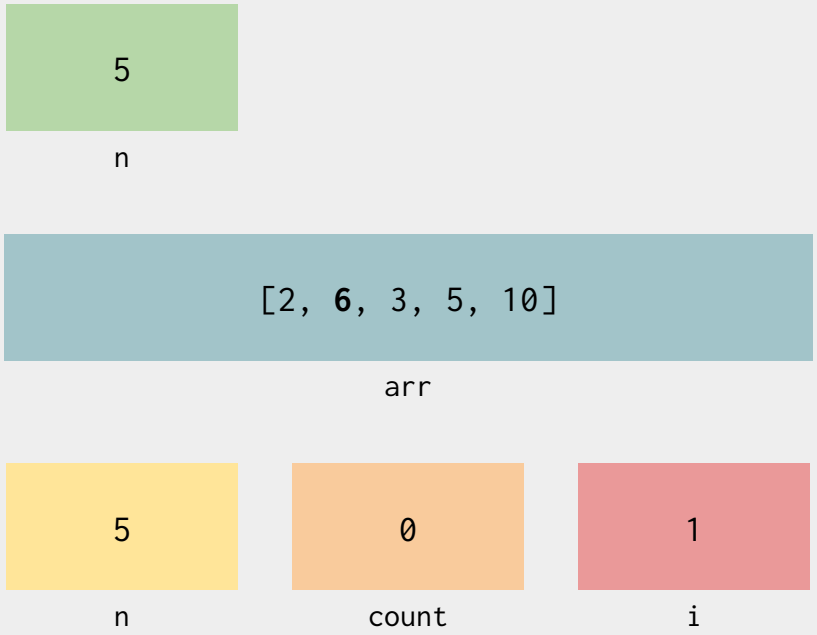
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



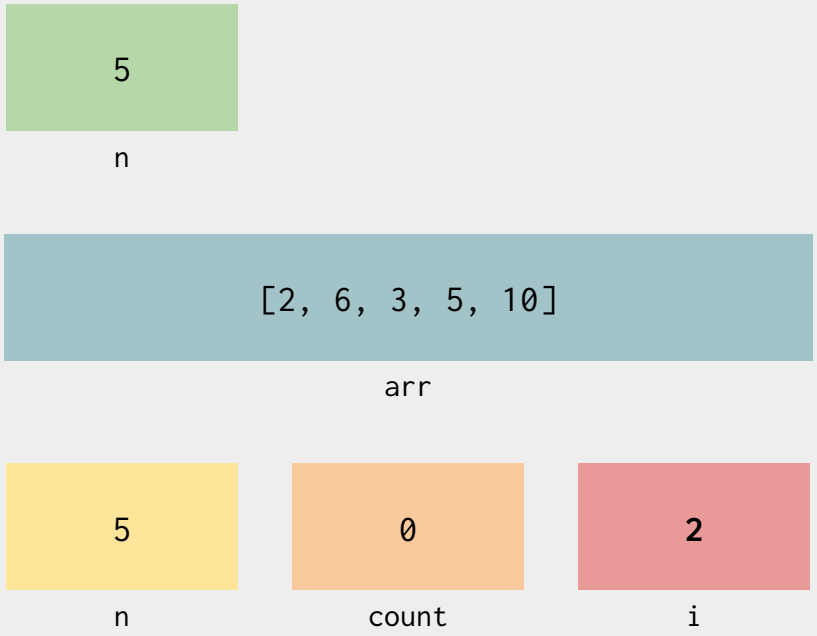
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



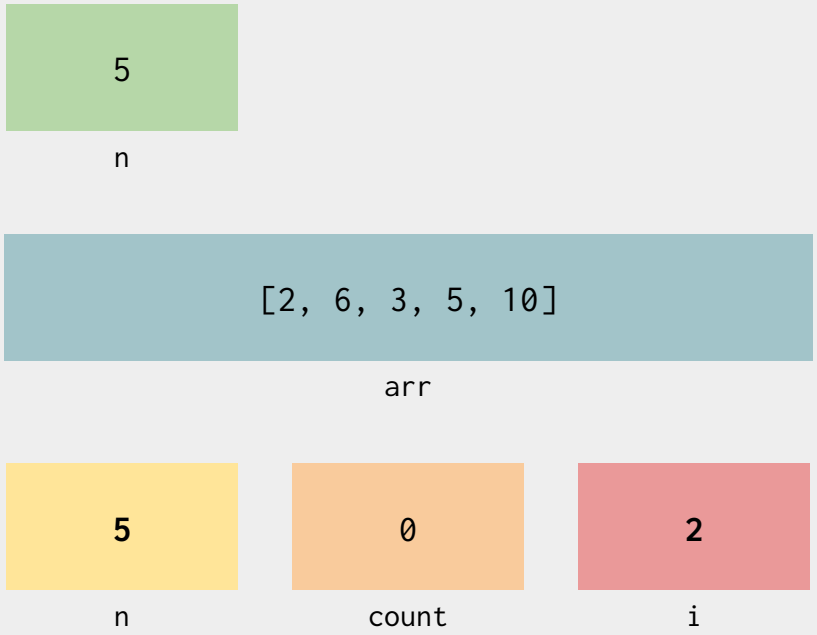
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



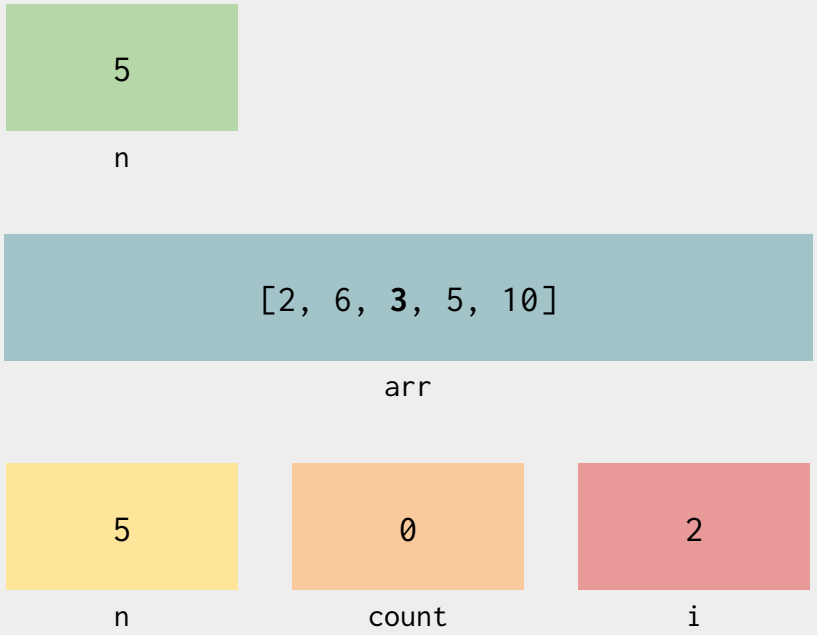
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



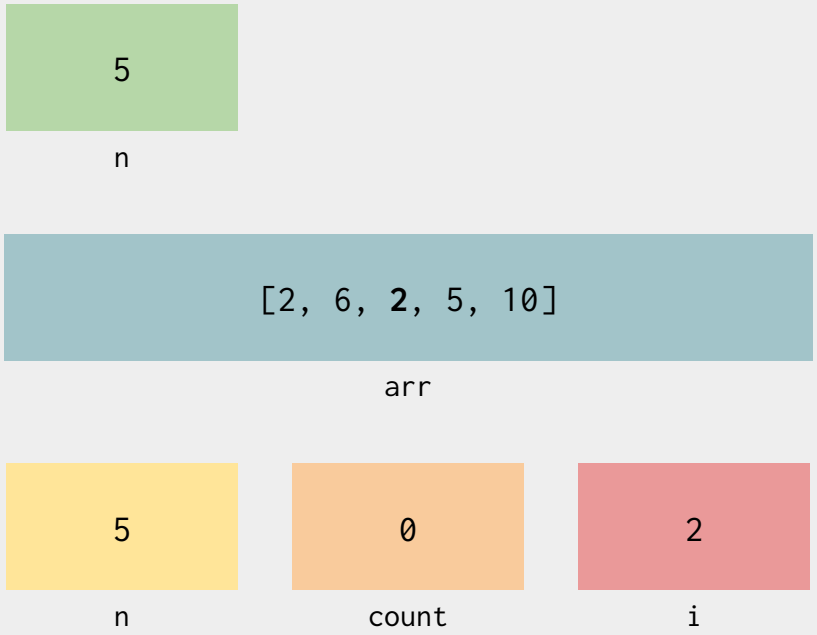
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



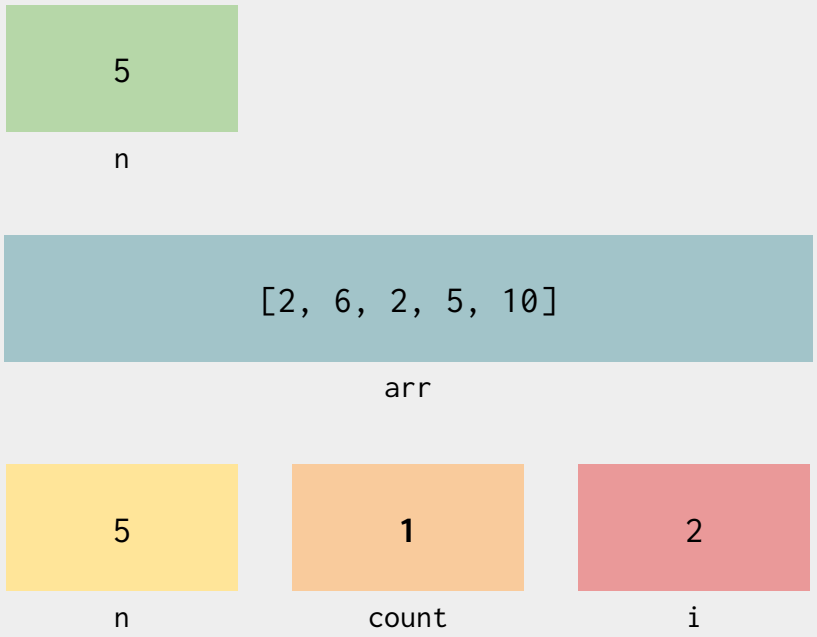
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



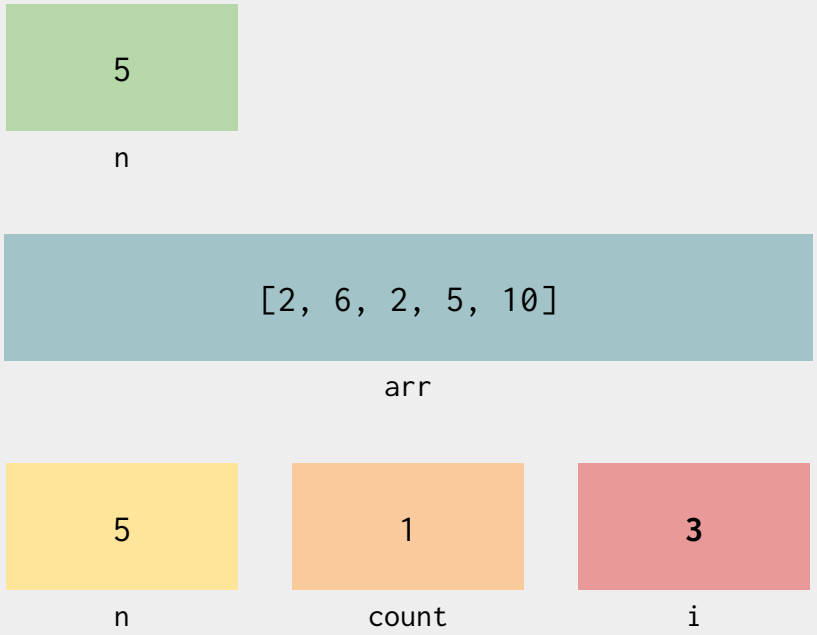
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



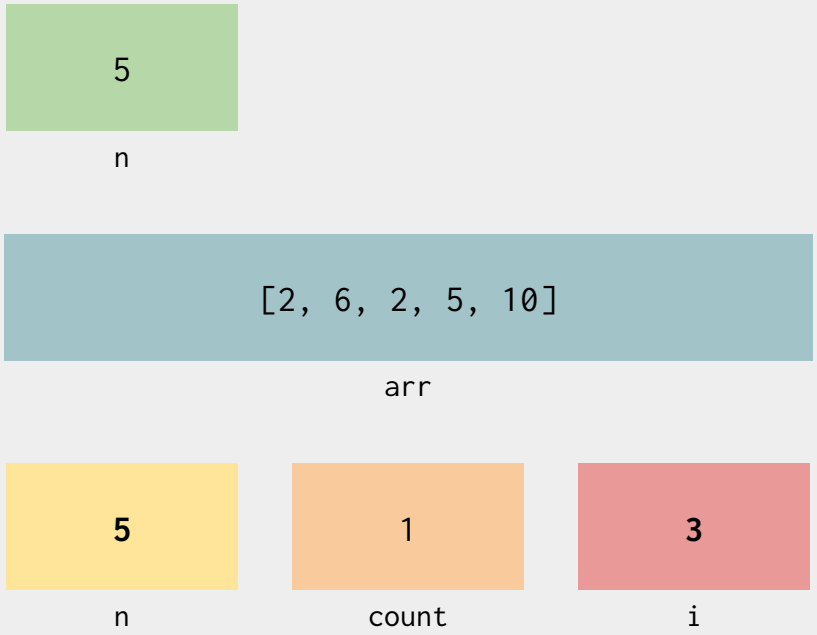
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



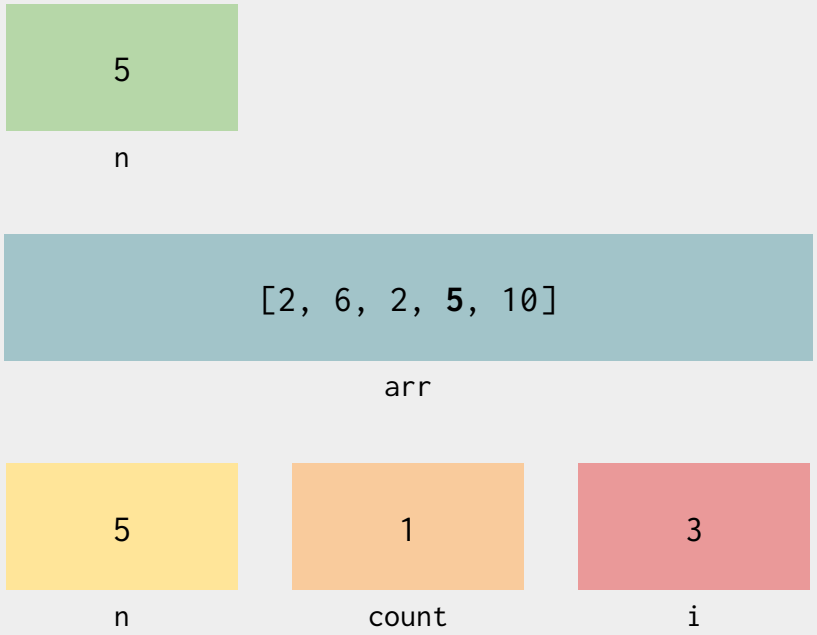
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



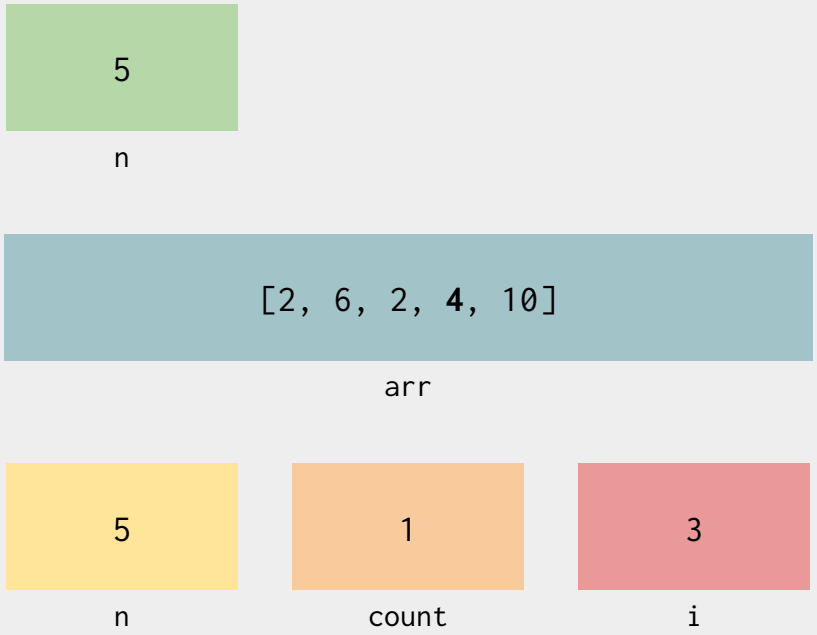
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



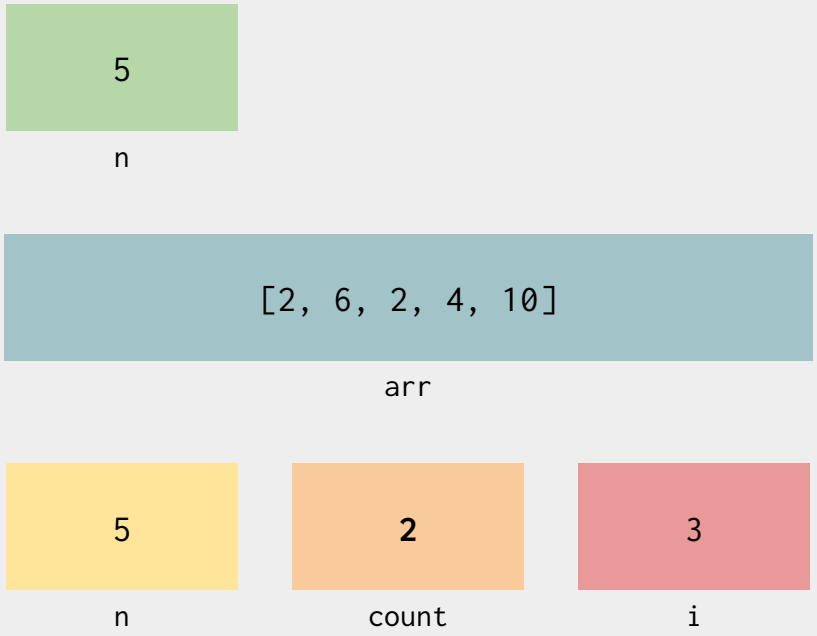
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



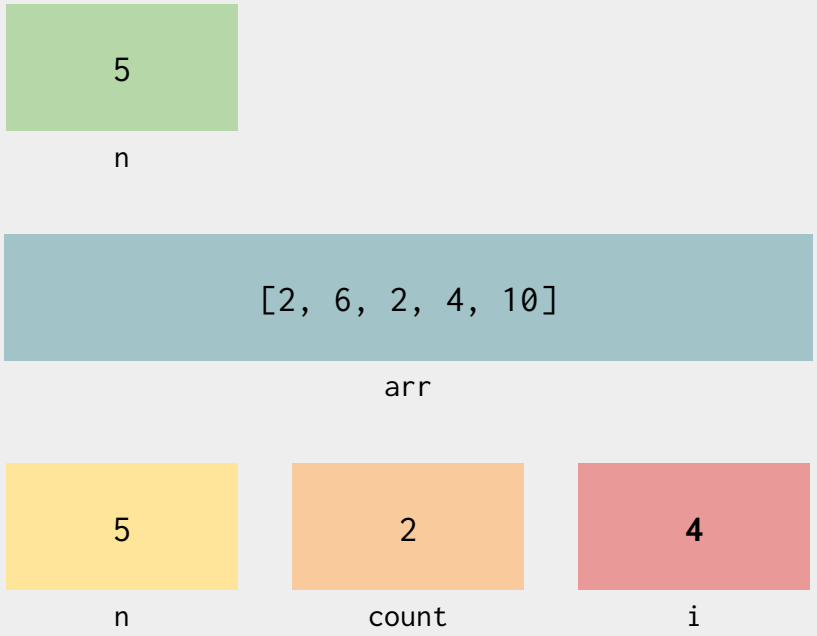
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



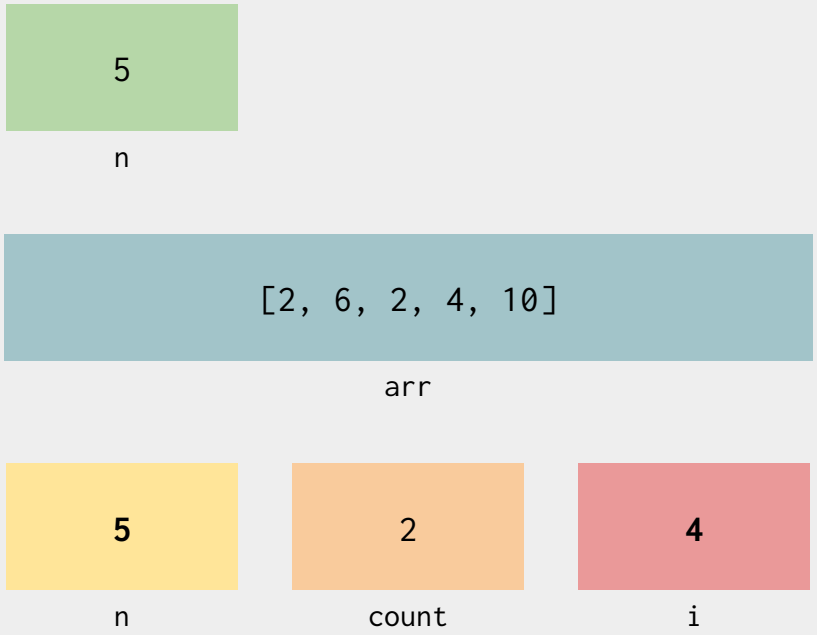
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



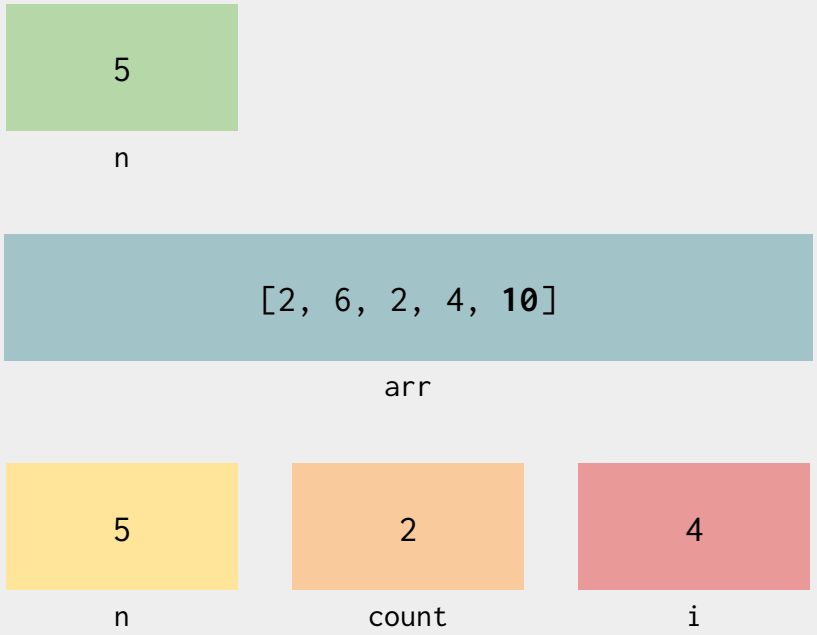
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



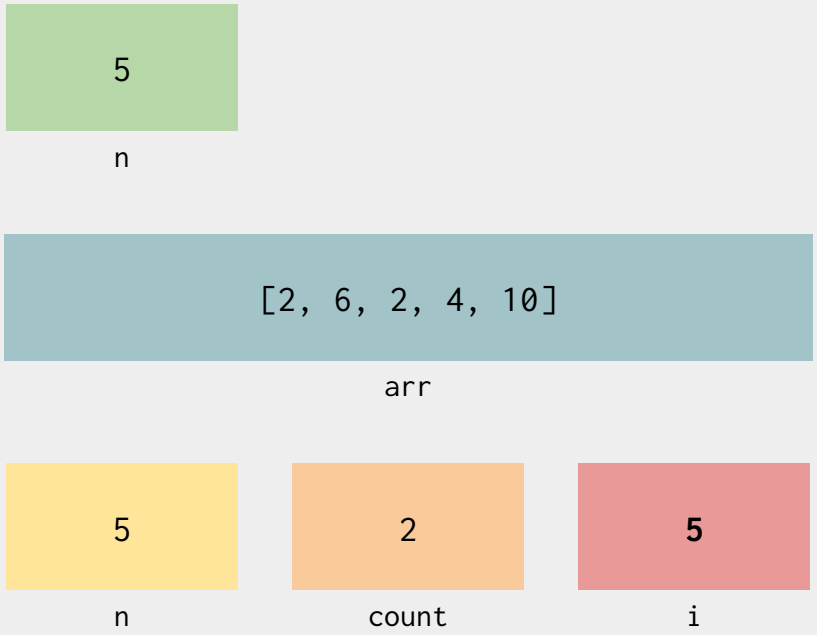
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



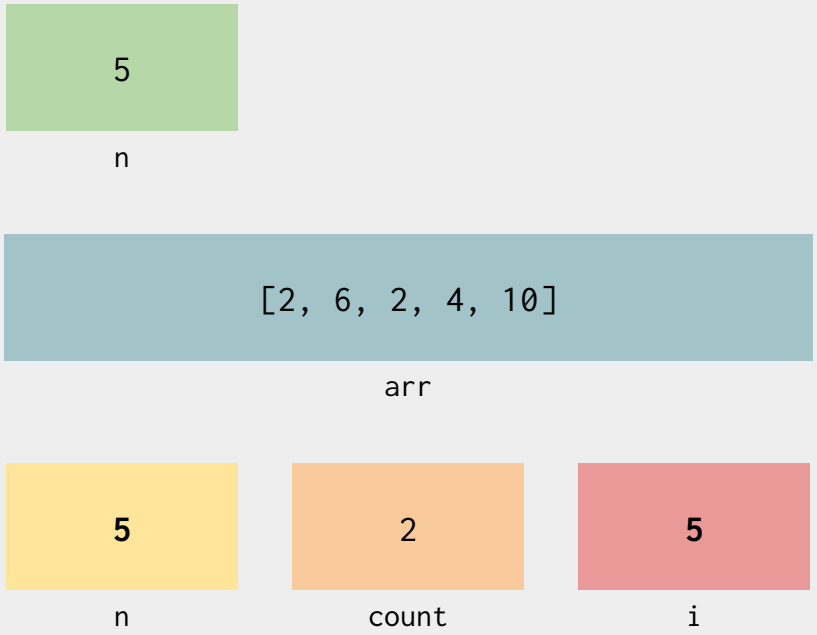
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



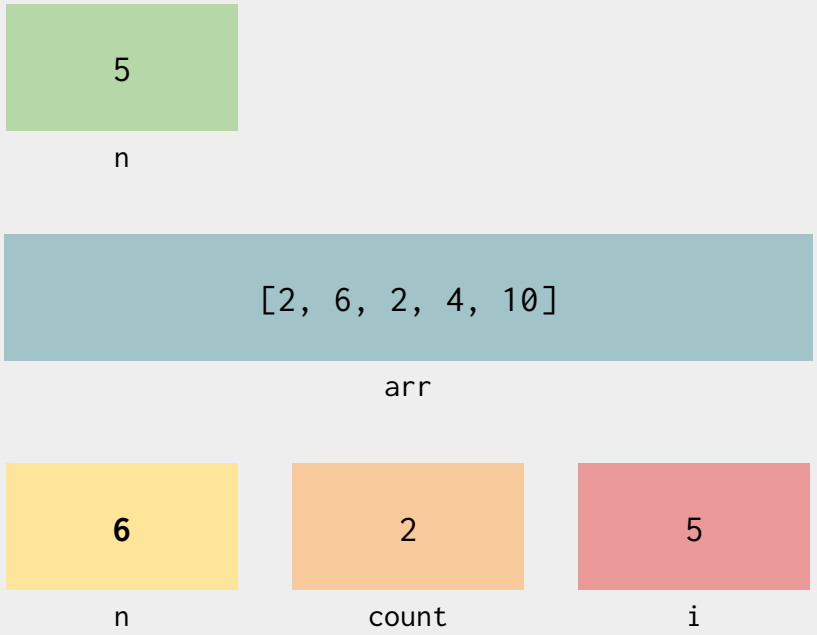
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



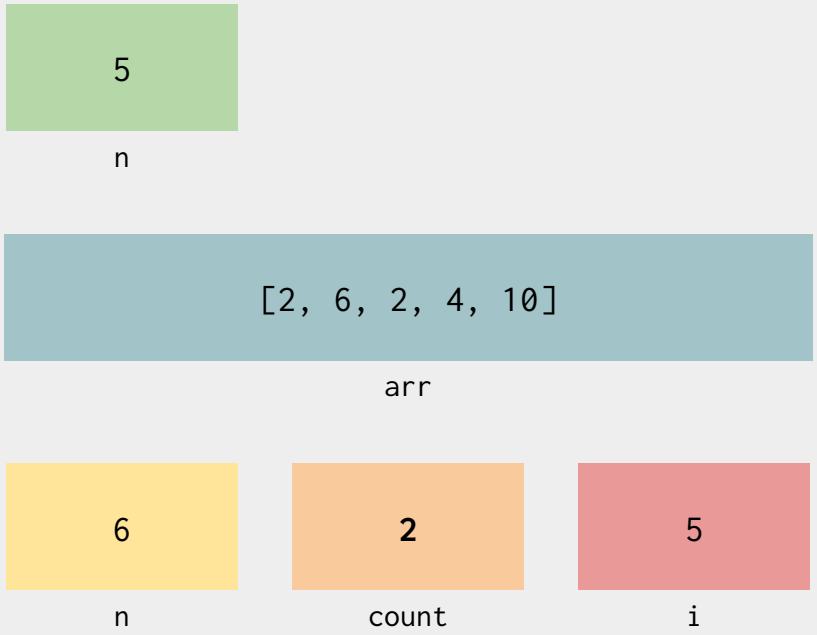
What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



What does this print?

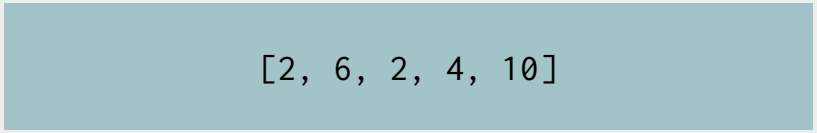
```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```



What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```

n

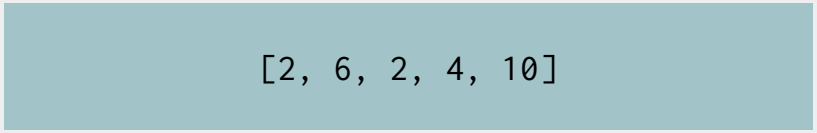


arr

What does this print?

```
// arr is the array itself, n is the size.
int changeOdd(int arr[], int n) {
  int count = 0;
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1) {
      arr[i]--;
      count++;
    }
  }
  n++;
  return count;
}
int main() {
      int n = 5;
      int arr[5] = {2, 6, 3, 5, 10};
      cout << changeOdd(arr, n) << endl;
}
```

n



arr

```
> 2
```

Printing Arrays

- To print an array, we need to use a loop to print each element.
- Printing the name will just print the starting address of the array

```
string arr[] = {“Smallberg”, “CS31”, “Midterm”};
for (int i = 0; i < 3; ++i) {
  cout << arr[i];
}
```

Out of Bounds Errors

- Occur anytime you can access memory past the end (or beginning) of an array
  - Only certain spaces in memory have useful data
  - Anything outside is essentially garbage
  - Hard to debug. C++ doesn't do bounds checking on array access so out of bounds accesses can often go unnoticed.

```
string array[3] = {“CS31”, “Smallberg”, “Midterm”};
cout << array[3] << endl; // Out of bounds error!
```

Out of Bounds Example

Do we have an out of bounds memory access here?

```
// Assume arr only contains n elements.
int countFives(int arr[], int n) {
  int count = 0;
  for (int i = 0; i <= n; ++i) {
    if (arr[i] == 5) {
      count++;
    }
  }
  return count;
}
```

Out of Bounds Example

Do we have an out of bounds memory access here?

```
// Assume arr only contains n elements
int countFives(int arr[], int n) {
  int count = 0;
  for (int i = 0; i <= n; ++i) {
    if (arr[i] == 5) {
      count++;
    }
  }
  return count;
}
```

Yes! The for loop will access the element at the **nth** index.

Practice Question: Index of First Repeated

Given an array of integers and the size of the array, write a function firstRepeat that returns the index of the first repeat element. Return -1 if there are no duplicate elements.

Input: `int arr[] = {1, 2, 3, 2, 4}; int size = 5;`
Output: `3`

Input: `int arr[] = {1, 2, 3, 7, 0, 2, 7, 3, 1}; int size = 9;`
Output: `5`

(Contributed by Carter Wu)

Solution: Index of First Repeated

We use nested for loops:

```
int firstRepeat(int a[], int n) {
    for (int i = 1; i < n; i++)
        for (int j = 0; j < i; j++)
            if (a[i] == a[j])
                return i;
    return -1;
}
```

Practice Question: What Makes CS Beautiful

```
int main() {
    string oneD[] = {"Zayn", "Louis", "Harry", "Niall", "Liam"};
    int size = 5;

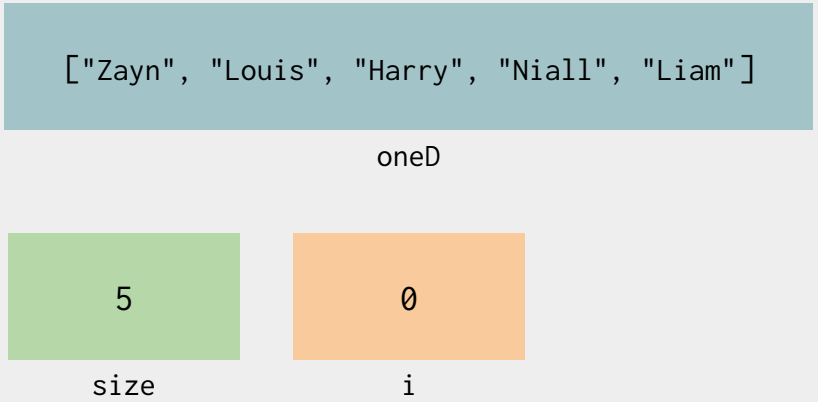
    for (int i = 0; i < size; i++) {
        int min = i;
        for (int j = i + 1; j < size; j++)
            if (oneD[j] < oneD[min])
                min = j;
        string temp = oneD[i];
        oneD[i] = oneD[min];
        oneD[min] = temp;
    }
    oneD[4] = "RIP" + oneD[4];
}
```

What does the string array contain after this code is executed?

Walkthrough

```
string oneD[] = {....};
int size = 5;

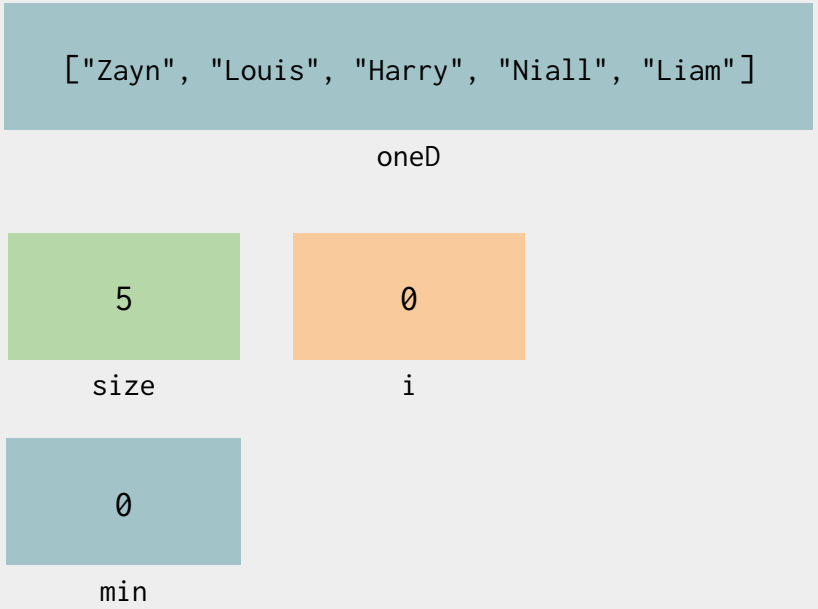
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

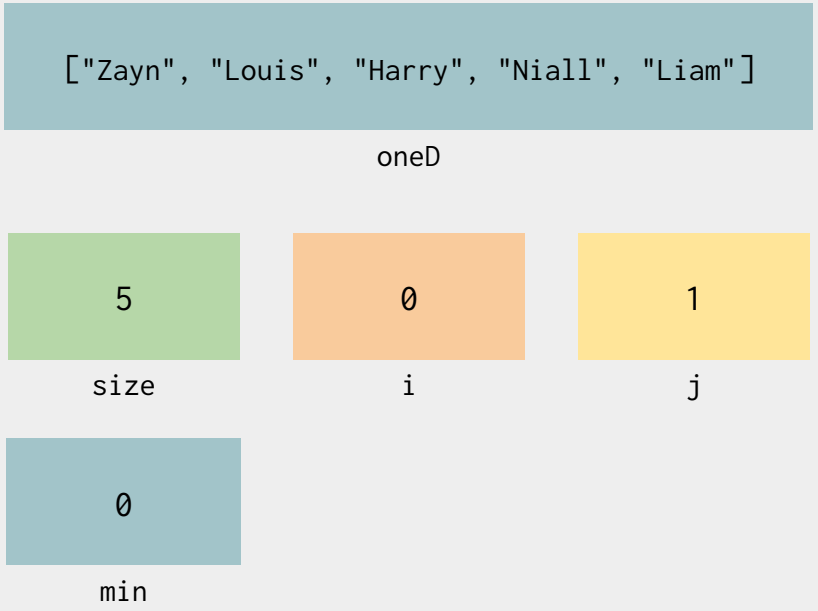
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

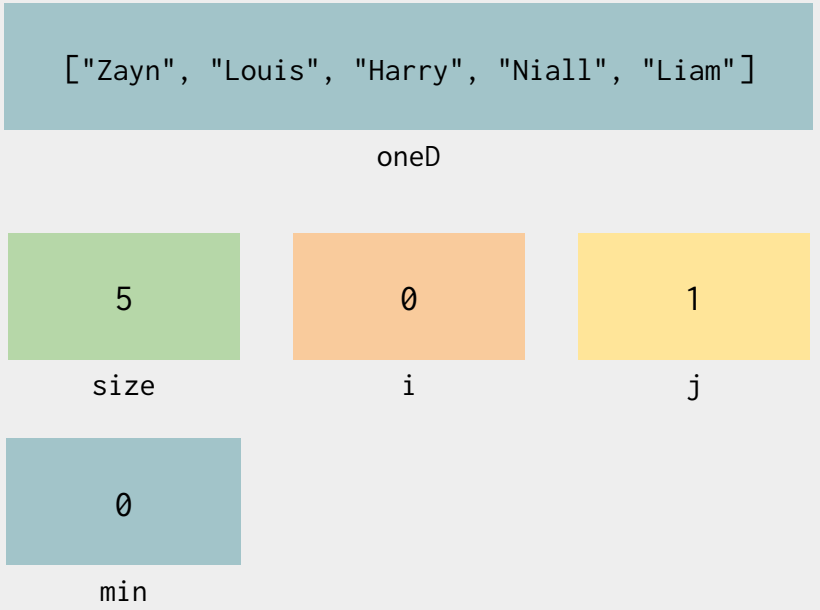
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

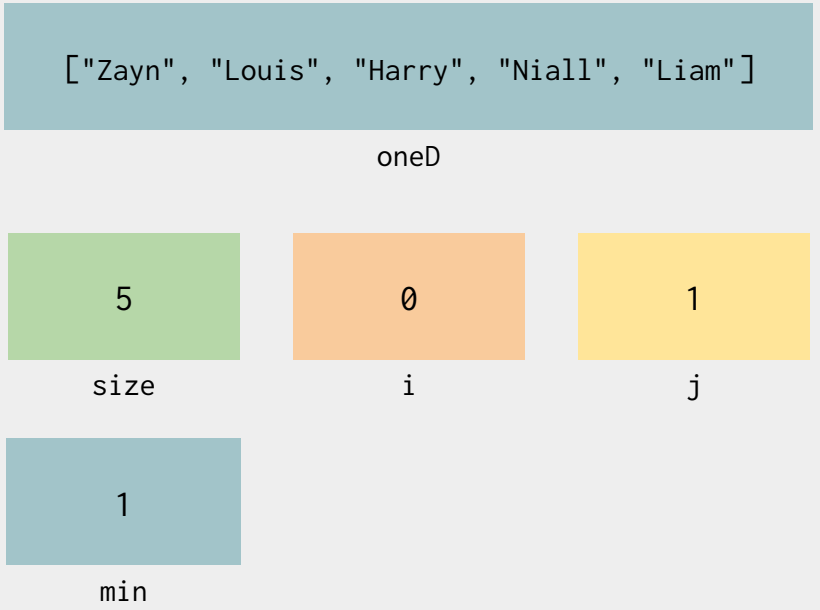
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

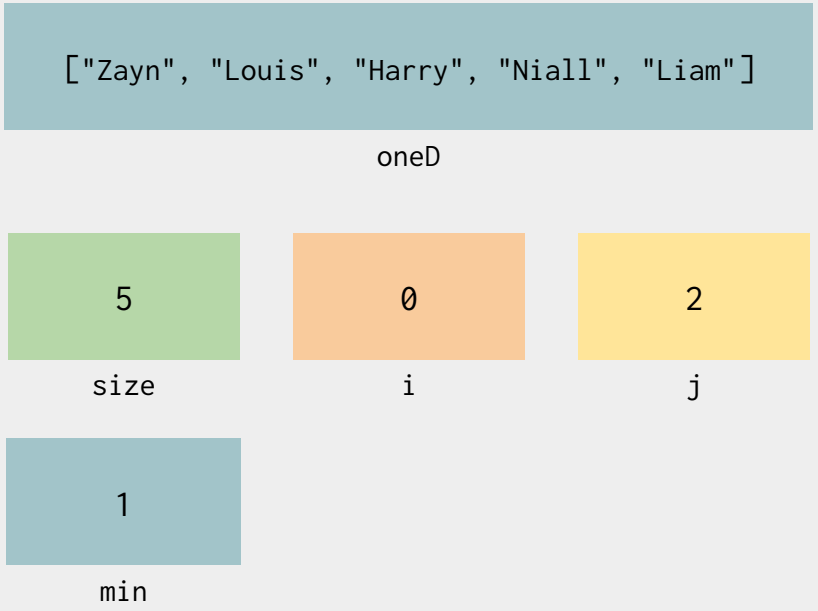
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

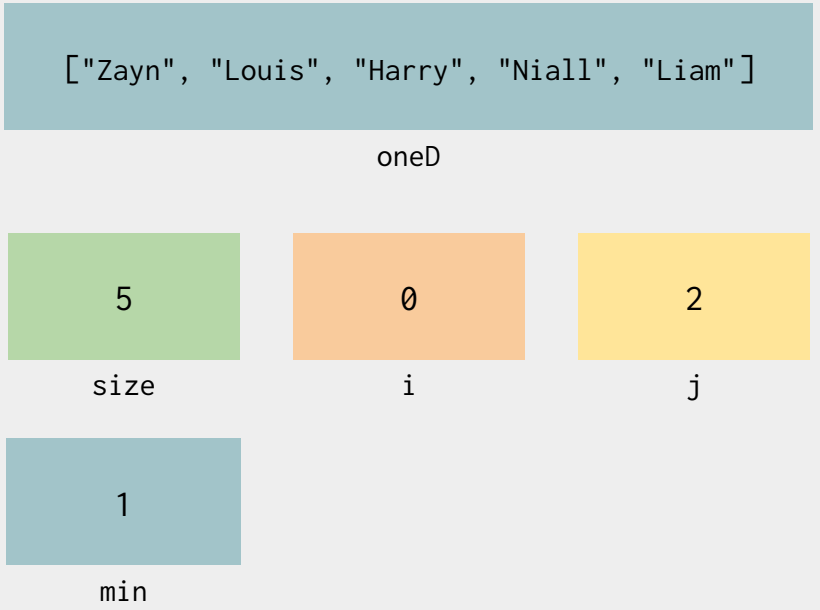
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

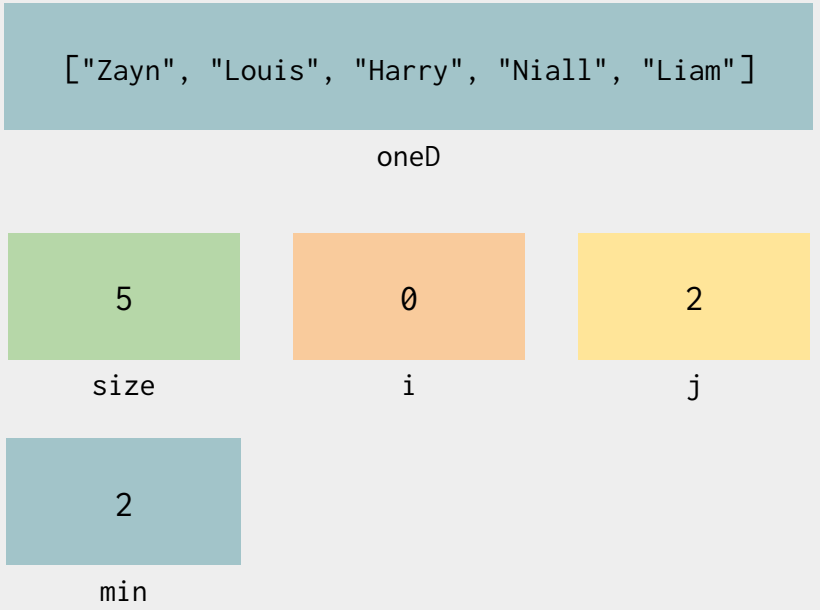
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

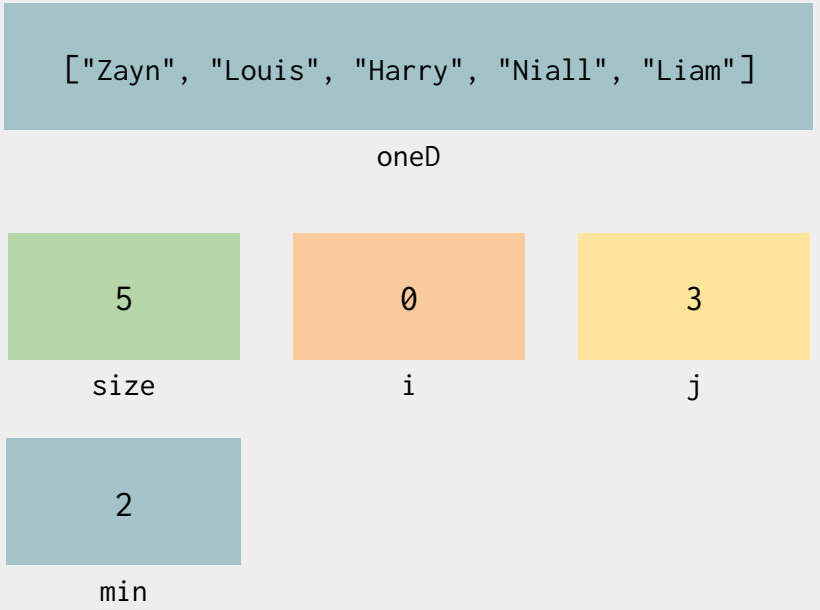
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

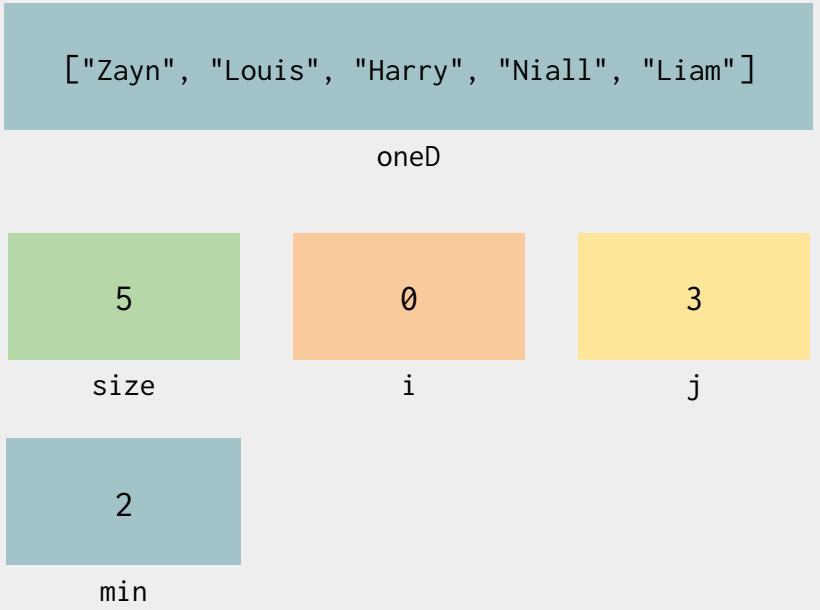
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

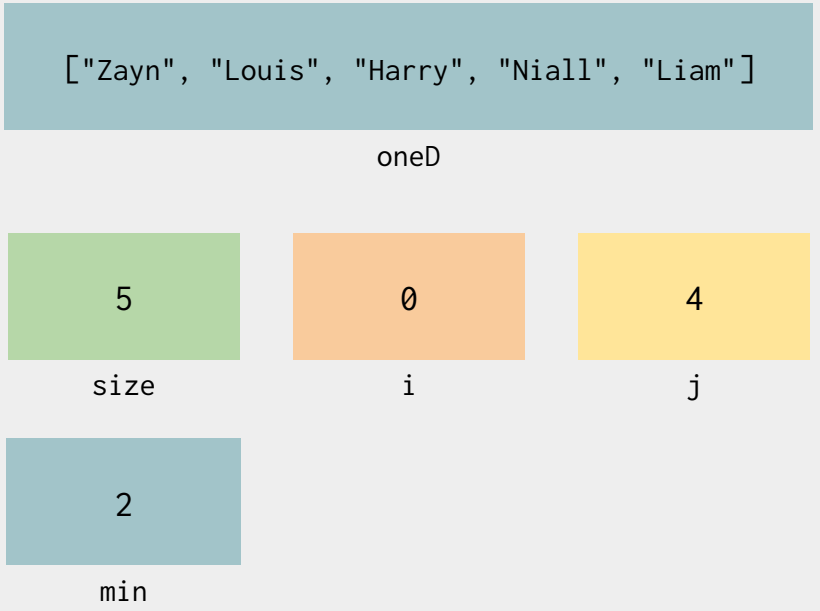
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

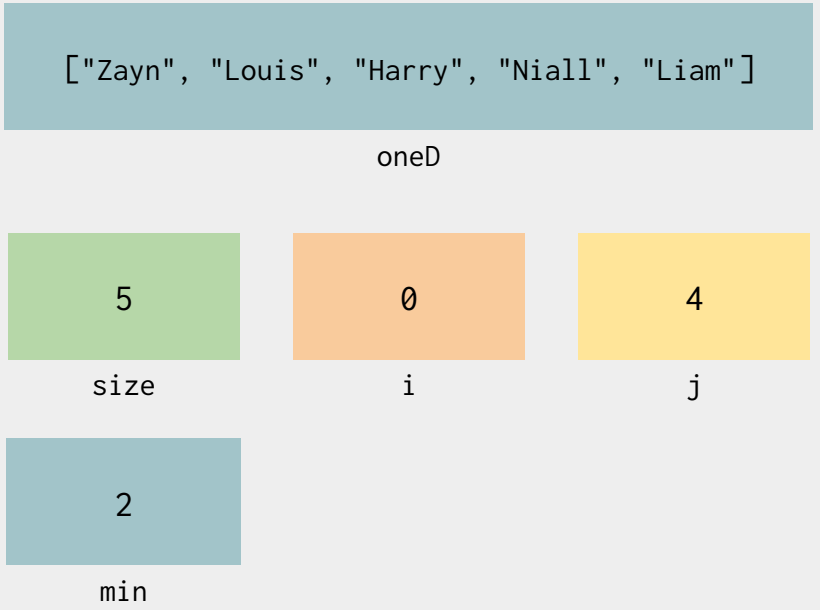
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

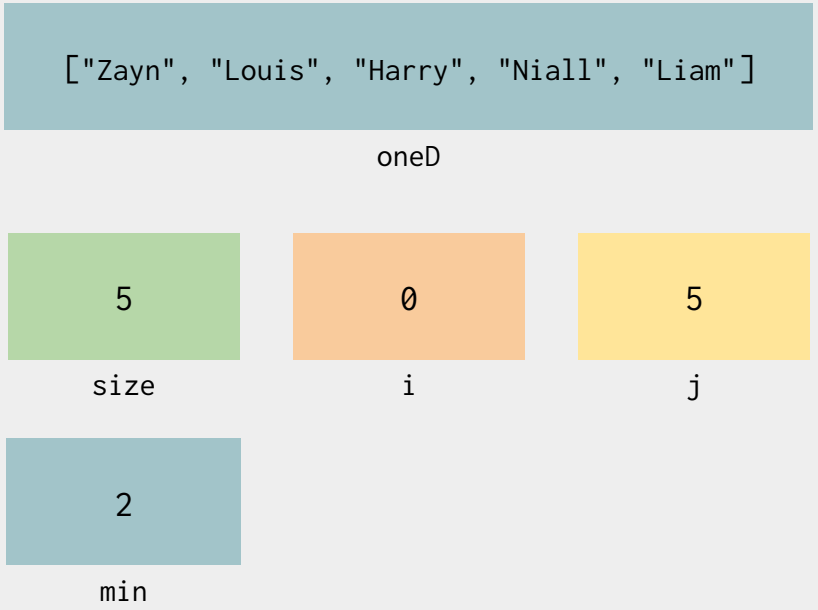
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

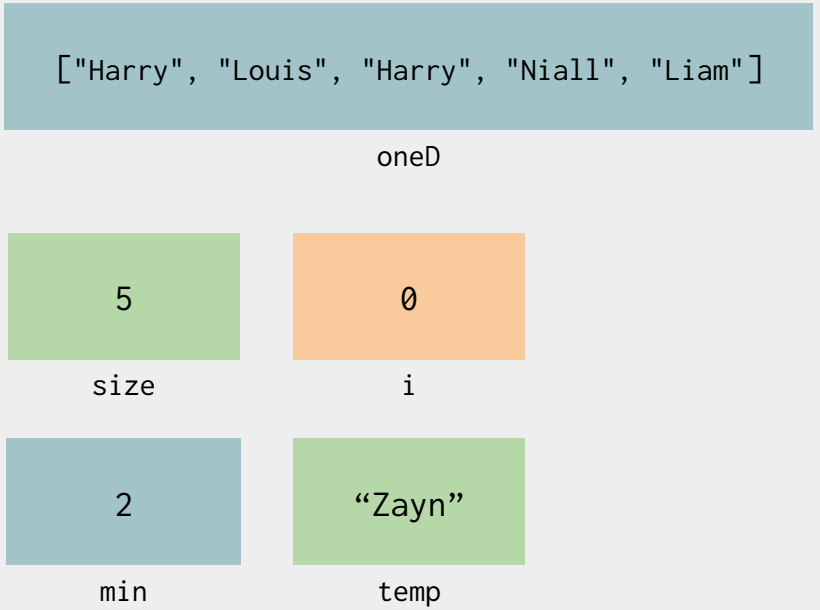
```
string oneD[] = {....};
int size = 5;

for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```

Walkthrough

```
string oneD[] = {....};
int size = 5;

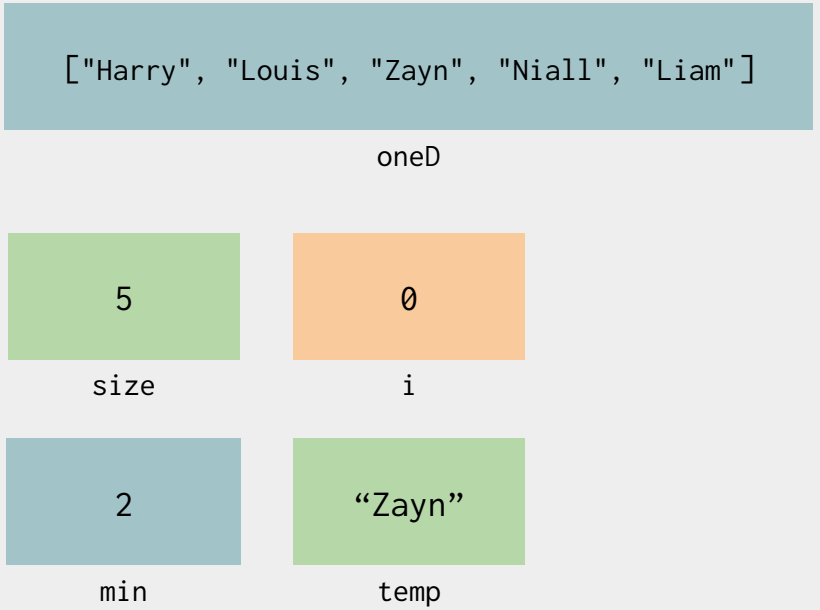
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

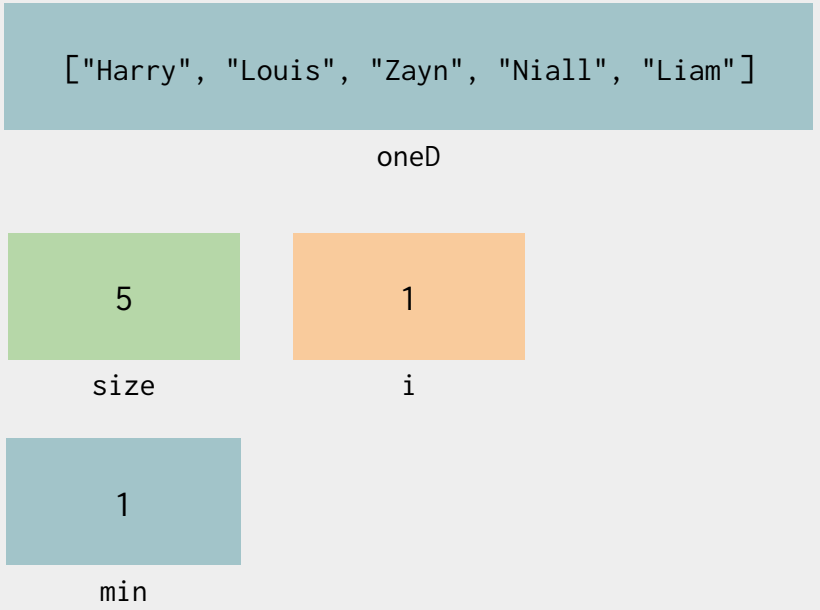
```
string oneD[] = {....};
int size = 5;

for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```

Walkthrough

```
string oneD[] = {....};
int size = 5;

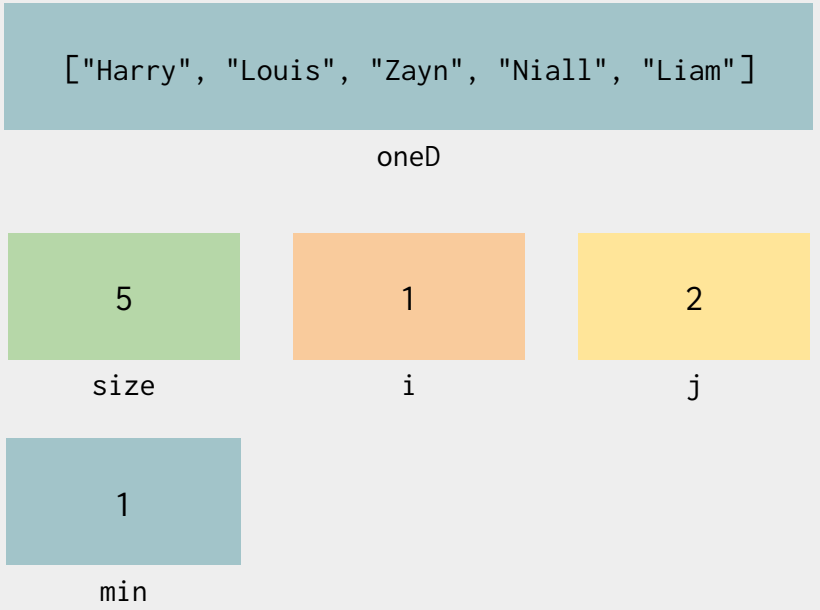
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

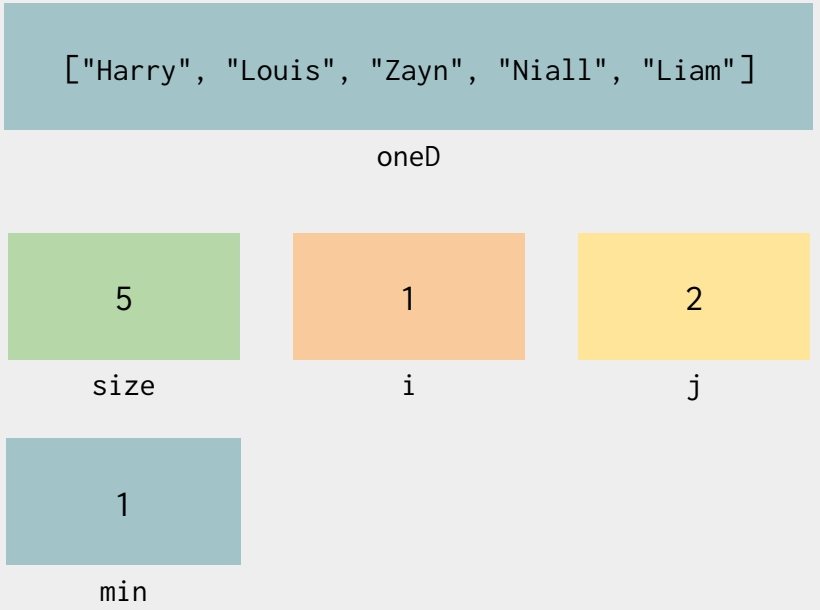
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

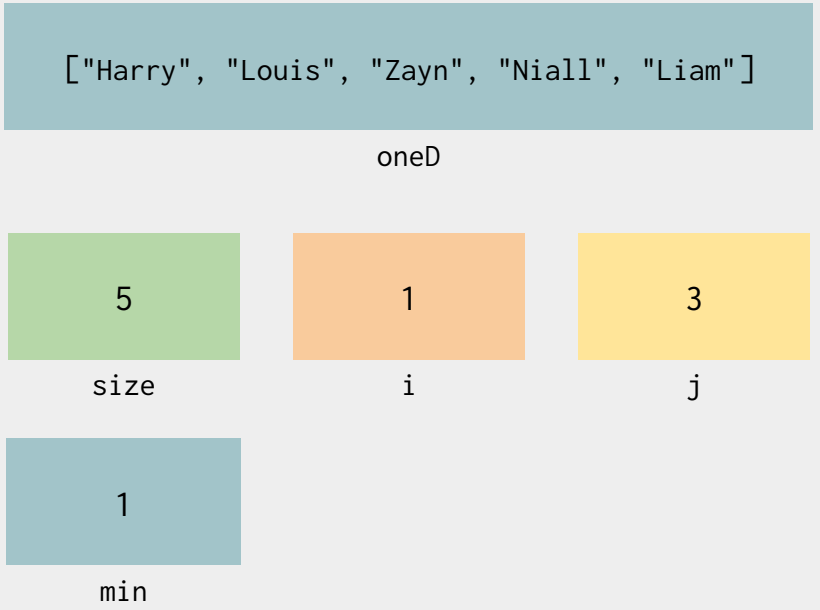
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

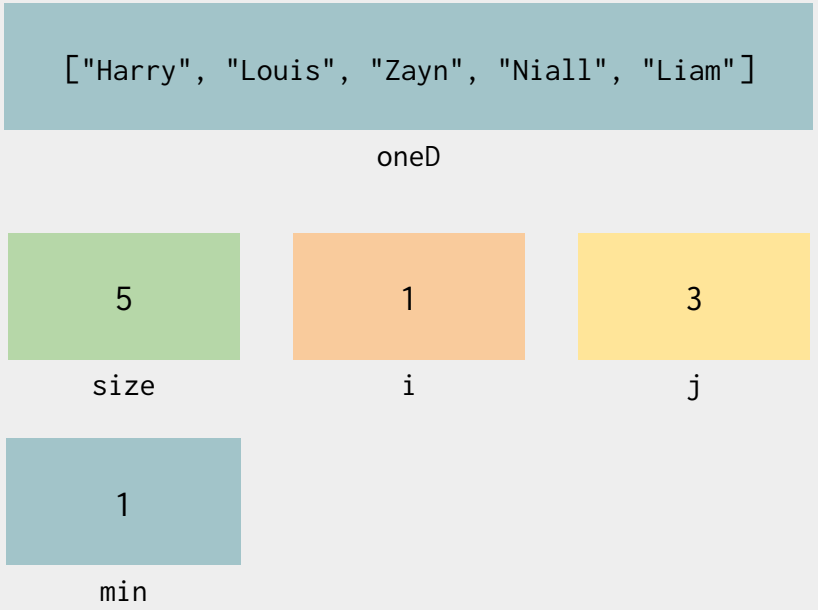
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

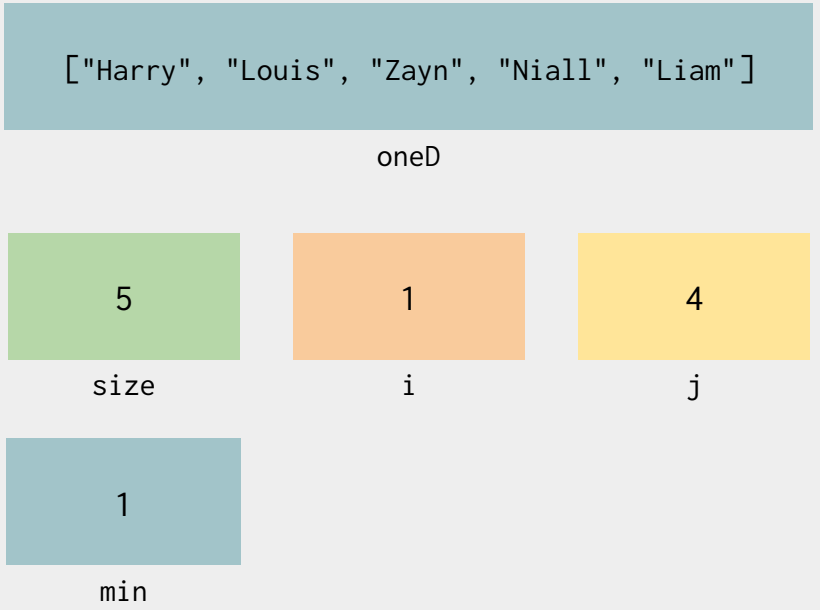
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

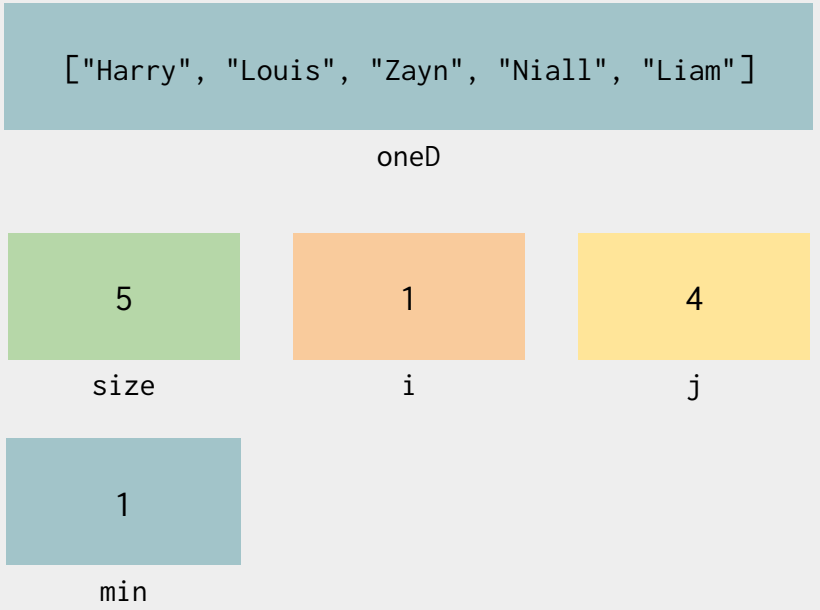
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

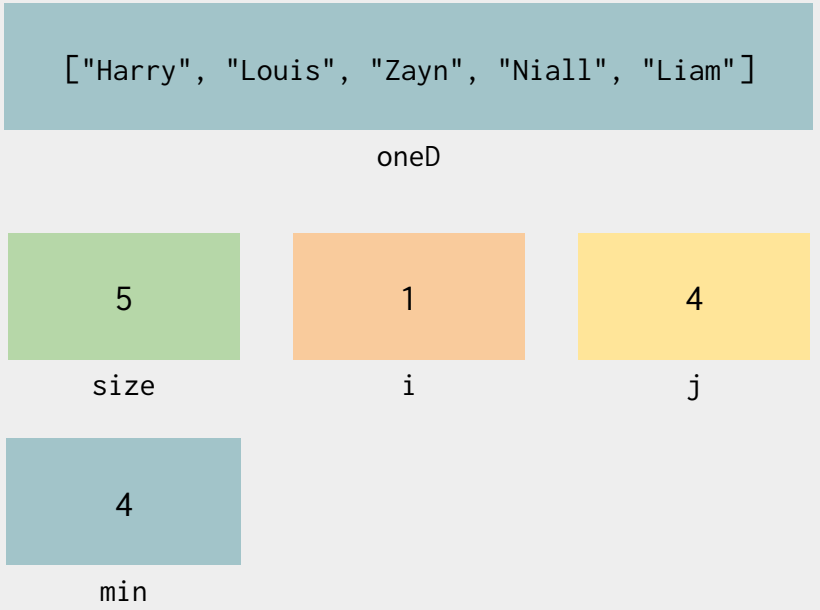
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

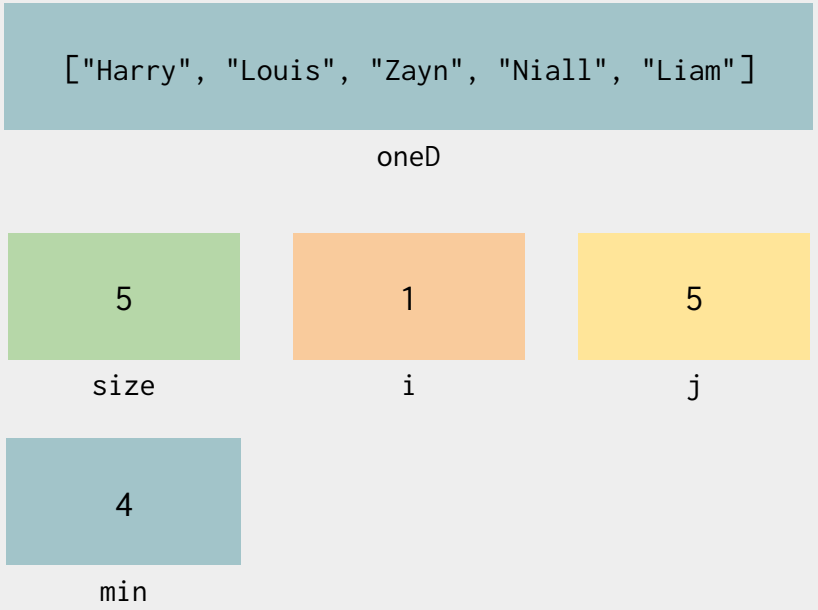
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

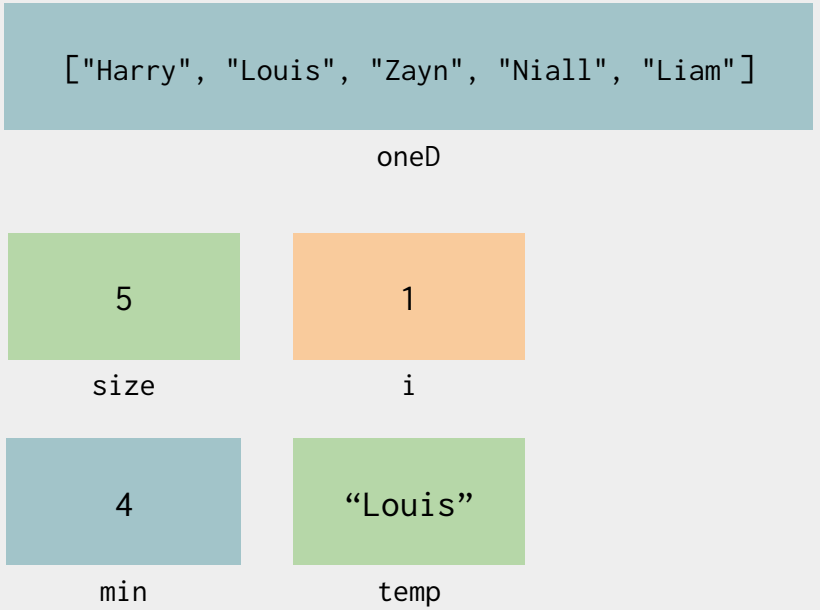
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

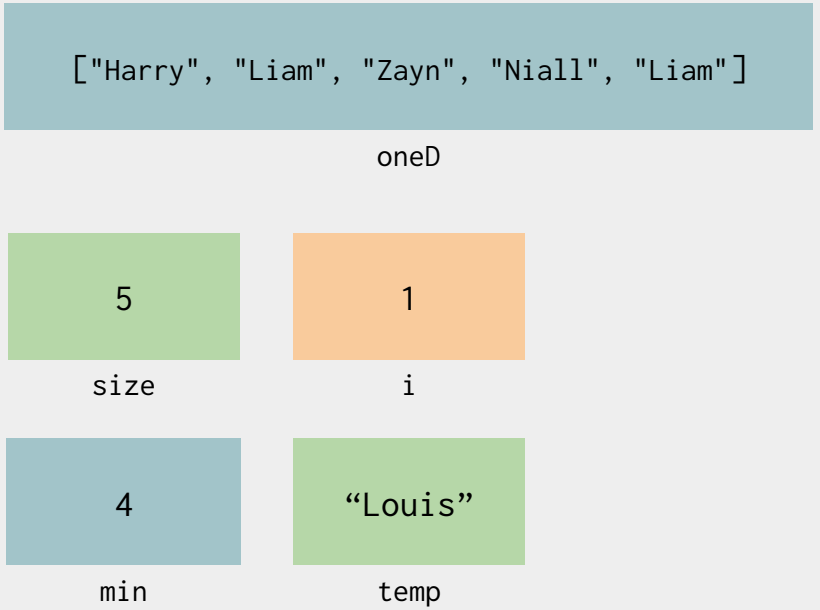
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

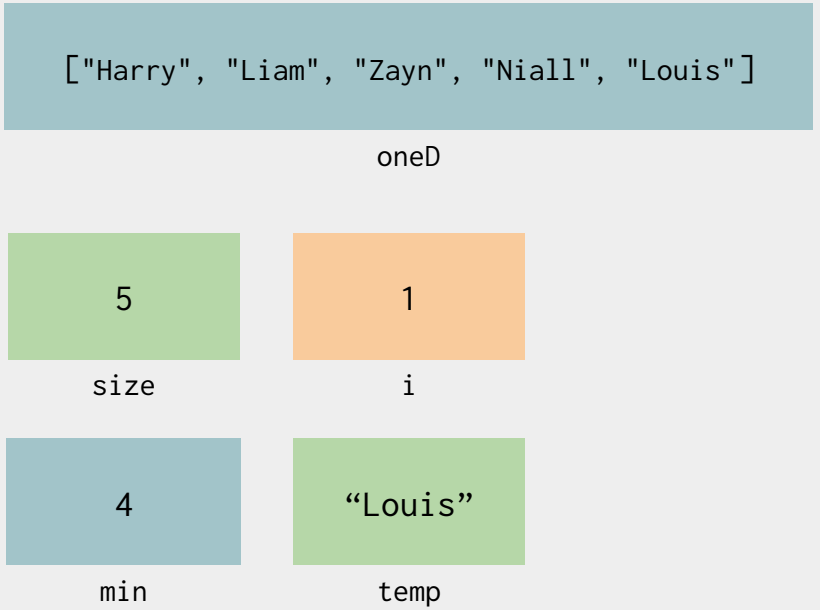
for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Walkthrough

```
string oneD[] = {....};
int size = 5;

for (int i = 0; i < size; i++) {
    int min = i;
    for (int j = i + 1; j < size; j++)
        if (oneD[j] < oneD[min])
            min = j;
    string temp = oneD[i];
    oneD[i] = oneD[min];
    oneD[min] = temp;
}
oneD[4] = "RIP" + oneD[4];
```



Solution: What Makes CS Beautiful

After walking through two iterations of the outer for loop, we notice that the loops are sorting the array into alphabetical order!

(this is called Selection Sort, but don't worry about it for now) https://en.wikipedia.org/wiki/Selection_sort

Initial: `["Zayn", "Louis", "Harry", "Niall", "Liam"]`

i = 0: `["Harry", "Louis", "Zayn", "Niall", "Liam"]`

i = 1: `["Harry", "Liam", "Zayn", "Niall", "Louis"]`

i = 2: `["Harry", "Liam", "Louis", "Niall", "Zayn"]`

i = 3: `["Harry", "Liam", "Louis", "Niall", "Zayn"]`

i = 4: `["Harry", "Liam", "Louis", "Niall", "Zayn"]`

Final Answer: `["Harry", "Liam", "Louis", "Niall", "RIPZayn"]`

Practice Question: Resolve Merge Issues

```
// Assume arr1 and arr2 are ordered from least to
// greatest and have size n1 and n2, respectively.
// Also assume arr3 has size n1 + n2.
void merge(int arr1[], int n1, int arr2[], int n2,
           int arr3[]) {
  int i1 = 0, i2 = 0, i3 = 0;
  while (i3 < n1 + n2) {
    if (arr1[i1] < arr2[i2]) {
      arr3[i3] = arr1[i1];
      i1++;
    } else if (arr2[i2] < arr1[i1]) {
      arr3[i3] = arr2[i2];
      i2++;
    }
    i3++;
  }
}
```

This function attempts to merge two arrays arr1 and arr2 that are ordered from least to greatest into a third array arr3, so that arr3 contains the contents of both arr1 and arr2 ordered from least to greatest.

Example: arr1 = {1, 2, 5}, arr2 = {2, 4, 6}
➔ arr3 = {1, 2, 2, 4, 5, 6}

Can you find and fix the bugs in this function so that it performs correctly?

Practice Question: Resolve Merge Issues

```
// Assume arr1 and arr2 are ordered from least to
// greatest and have size n1 and n2, respectively.
// Also assume arr3 has size n1 + n2.
void merge(int arr1[], int n1, int arr2[], int n2,
           int arr3[]) {
  int i1 = 0, i2 = 0, i3 = 0;
  while (i3 < n1 + n2) {
    if (arr1[i1] < arr2[i2]) {  // what if i1>=n1
      arr3[i3] = arr1[i1];      // or i2 >= n2??
      i1++;
    } else if (arr2[i2] < arr1[i1]) {  // same!
      arr3[i3] = arr2[i2];
      i2++;
    }  // what do we do if arr1[i1] == arr2[i2]?
    i3++;
  }
}
```

This function attempts to merge two arrays arr1 and arr2 that are ordered from least to greatest into a third array arr3, so that arr3 contains the contents of both arr1 and arr2 ordered from least to greatest.

Example: arr1 = {1, 2, 5}, arr2 = {2, 4, 6}
➔ arr3 = {1, 2, 2, 4, 5, 6}

Can you find and fix the bugs in this function so that it performs correctly?

Solution: Resolve Merge Issues

```
void merge(int arr1[], int n1, int arr2[], int n2,
           int arr3[]) {
  int i1 = 0, i2 = 0, i3 = 0;
  while (i1 < n1 && i2 < n2) {
    if (arr1[i1] < arr2[i2]) {
      arr3[i3] = arr1[i1];
      i1++;
    } else if (arr2[i2] <= arr1[i1]) {
      arr3[i3] = arr2[i2];
      i2++;
    }
    i3++;
  }
  // continued...
  while (i1 < n1) {  // only one of these will run
    arr3[i3] = arr1[i1];
    i1++;
    i3++;
  }
  while (i2 < n2) {
    arr3[i3] = arr2[i2];
    i2++;
    i3++;
  }
}
```

C Strings

C Strings

- C does not have the string class *(or classes at all!)*
- In C, we cannot declare strings or use class methods:
  - `string x = “hello”;`
    - `x.size()` **`// This is okay in C++, but not in C.`**
- Instead, we represent strings using char arrays:
  - `char y[] = “hello”;`
  - Cannot use C++ string functions with it
    - `y.size(), y.substr(...), etc.` **`// Syntax errors.`**
  - **`#include <cstring>`** provides functions like **`strlen`**
    - **`strlen(x)`** returns 5

Ascii: Characters are actually integers

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Ascii Table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Ascii (example)

```
int x = ‘G’;
x -= 1;
char y = x;
int z = ‘5’;
```

Integer value (decimal)	char
70	‘F’
71	‘G’
53	‘5’

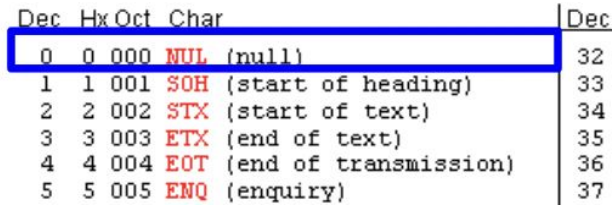
Ascii (example)

```
int x = ‘G’; // x is now 71.
x -= 1;      // x is now 70.
char y = x;  // y is now ‘F’.
int z = ‘5’; // z is now 53.
```

Integer value (decimal)	char
70	‘F’
71	‘G’
53	‘5’

C Strings (cont.)

- The end of a C string is marked by a zero byte (**‘\0’**)
  - Zero byte has ASCII value `0`
  - **`strlen`** simply looks for the zero byte for you



Dec	Hx	Oct	Char		Dec
0	0	000	NUL	(null)	32
1	1	001	SOH	(start of heading)	33
2	2	002	STX	(start of text)	34
3	3	003	ETX	(end of text)	35
4	4	004	EOT	(end of transmission)	36
5	5	005	ENQ	(enquiry)	37

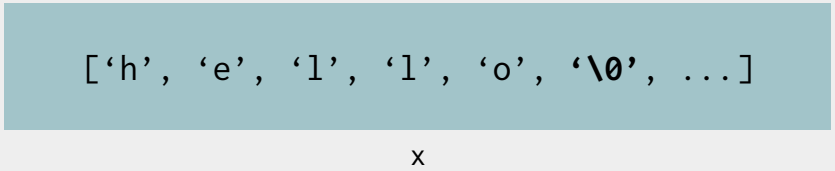
```
char arr[] = “hello”;
for (int i = 0; arr[i] != ‘\0’; i++){ // Standard for loop to iterate
…}                                    // through c-strings.
```

Note: `arr[i] != ‘\0’` and `arr[i] != 0` are the same, as ascii value of **‘\0’** is **0**.

C Strings (cont.)

```
// A zero byte is automatically
// put in index 5.
char x[50] = “hello”;

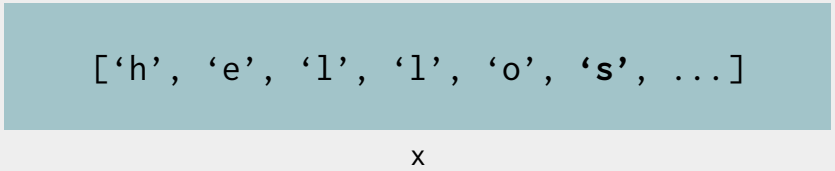
// Because we have more space in the array
// (50 total), we can add more characters.
x[5] = ‘s’;
x[6] = ‘\0’;
```



C Strings (cont.)

```
// A zero byte is automatically
// put in index 5.
char x[50] = “hello”;

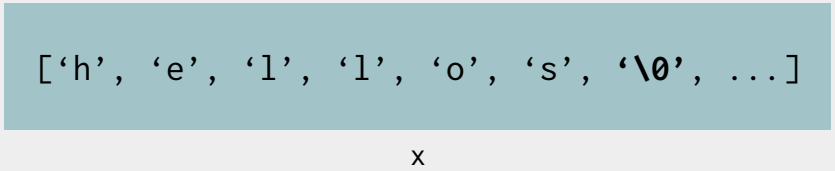
// Because we have more space in the array
// (50 total), we can add more characters.
x[5] = ‘s’;
x[6] = ‘\0’;
```



C Strings (cont.)

```
// A zero byte is automatically
// put in index 5.
char x[50] = “hello”;

// Because we have more space in the array
// (50 total), we can add more characters.
x[5] = ‘s’;
x[6] = ‘\0’;
```



Practice Question: C Strings - removeNonAlpha

Given a C String, write a function removeNonAlpha that removes all non-alphabet chars in the C String. When removing a non-alphabet char, you should shift all following chars one position to the left. Don't forget to shift the zero byte as well!

```
char cstr[] = "S5mal.lb-erg! Is+ C$s Senpai$$$";
removeNonAlpha(cstr);
for (int i = 0; cstr[i] != '\0'; i++)
    cout << cstr[i];

// OUTPUT: SmallbergIsCsSenpai
```

(Contributed by Matt Wong)

Solution: C Strings - removeNonAlpha

The outer for loop iterates through every character position in the C String. The inner while loop and for loop shifts the characters to the left to remove all non-alphabet characters.

```
#include <cctype> // for the use of isalpha() function

void removeNonAlpha(char str[]) {
    for(int i = 0; str[i] != '\0'; i++){ // iterate thru the cstring
        while ( !isalpha(str[i]) && str[i] != '\0' ){ // handle non-alpha char
            for(int j = i; str[j] != '\0'; j++)  // shift all the char after it left by 1
                str[j] = str[j+1];
        }
    }
}
```

Good luck!

Feedback	https://tinyurl.com/upefeedback
Sign-in	https://tinyurl.com/cs31mt2signin
Slides	https://tinyurl.com/cs31mt2slides
Practice	https://github.com/uclaupe-tutoring/practice-problems/wiki

Questions? Need more help?

- Come up and ask us! We'll try our best.
- UPE offers daily computer science tutoring:
  - Location: ACM/UPE Clubhouse (Boelter 2763)
  - Schedule: https://upe.seas.ucla.edu/tutoring/
- You can also post on the Facebook event page.