UPE Tutoring:

CS 31 Midterm 2 Review

Sign-in https://goo.gl/qSWgWN

Slides link available upon sign-in

Table of Contents

- Arrays
- C-Strings
- Pointers

Practice Questions:

- First Repeat Index
- One Direction Sort
- C-String Reversal
- Arrays w/ Pointers
- strcat

Arrays

- Valid declarations:

```
int arr[10];

bool list[5];

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

int arr[] = {1, 2, 3};
```

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

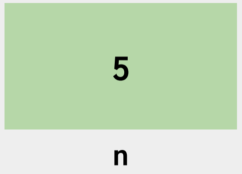
```
// arr is the array itself, n is the size.
int firstOdd(int arr[], int n) {
  for (int i = 0; i < n; i++) {
    if (arr[i] % 2 == 1)
      return i;
  }
  return n; // If no odd number found.
}
```

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;
}
```

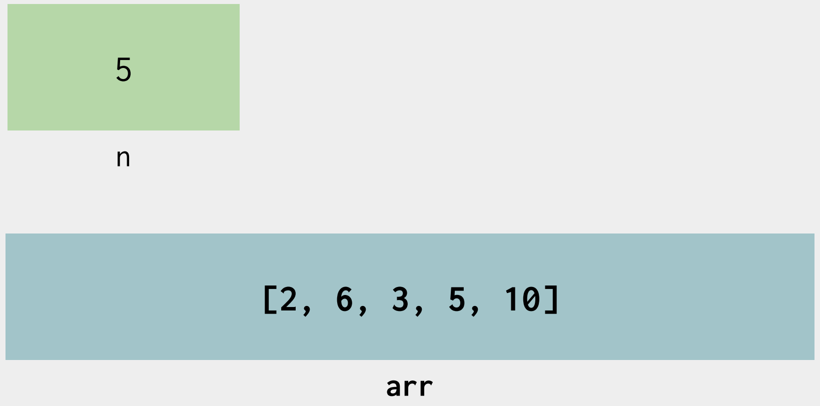
What does this print?

```
// 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;
}
```



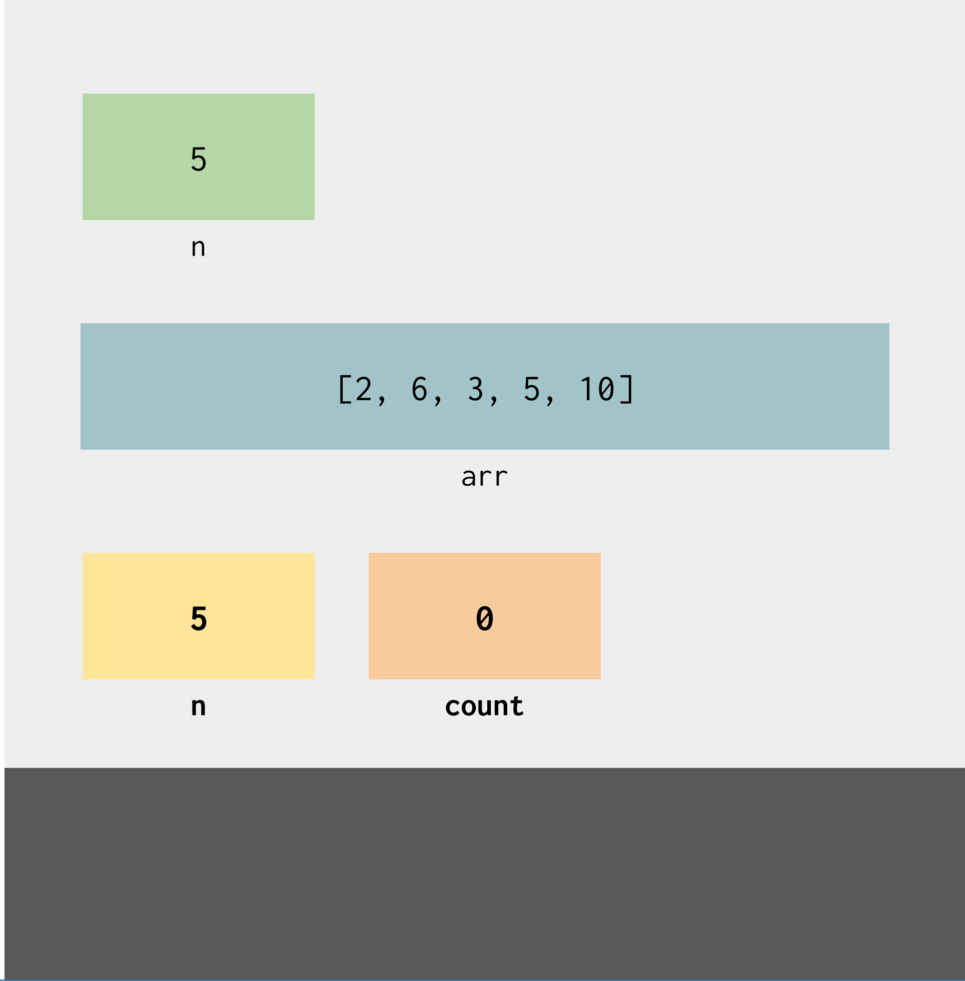
What does this print?

```
// 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;
}
```



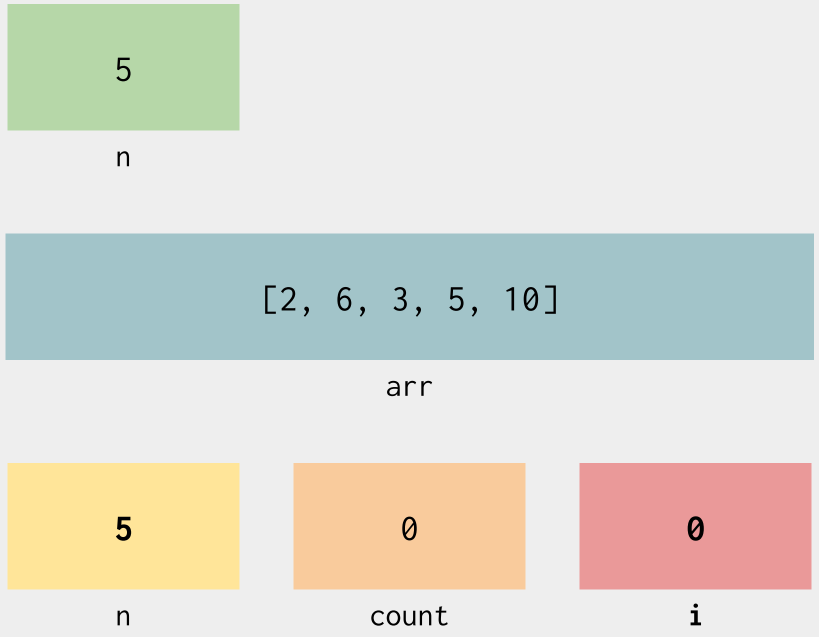
What does this print?

```
// 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;
}
```



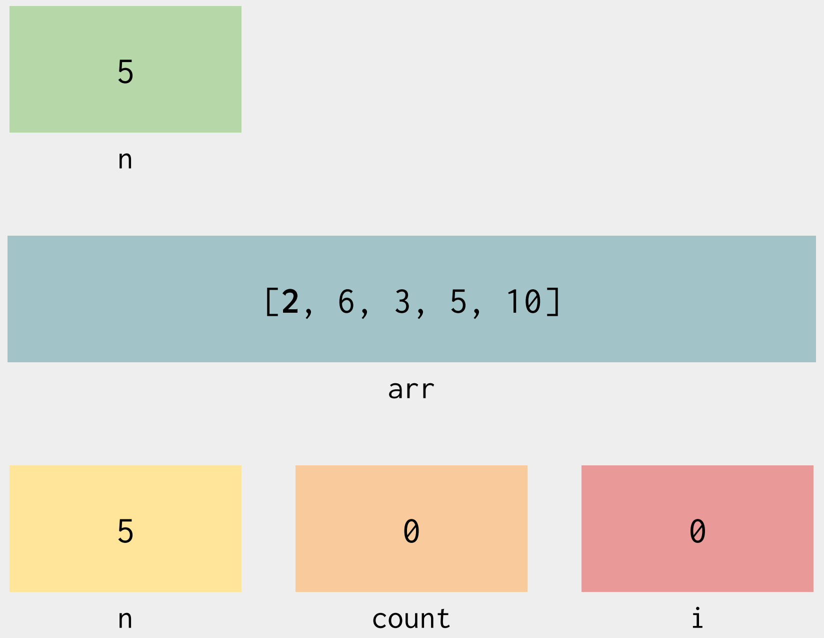
What does this print?

```
// 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;
}
```



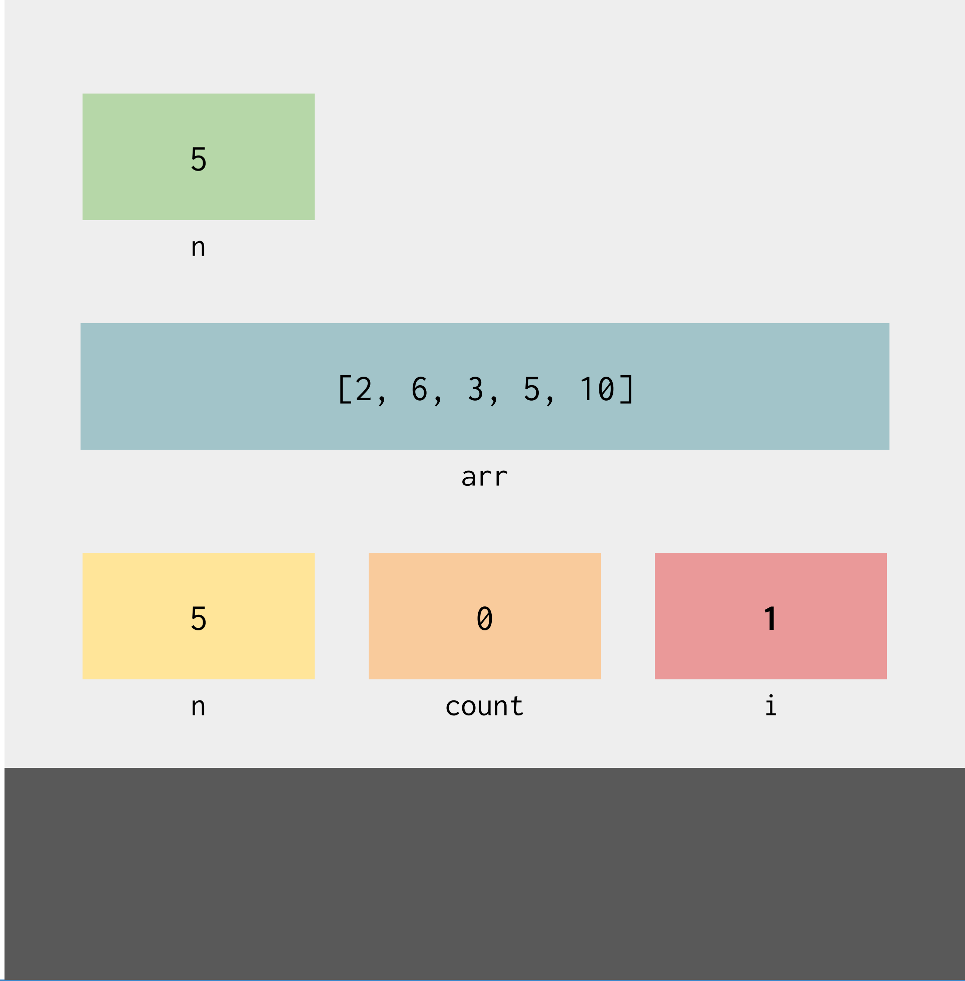
What does this print?

```
// 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;
}
```



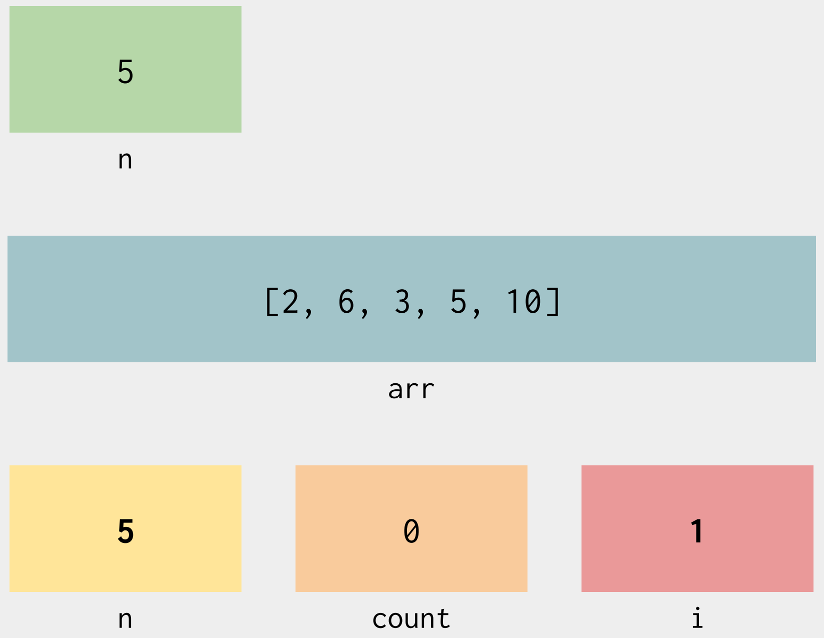
What does this print?

```
// 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;
}
```



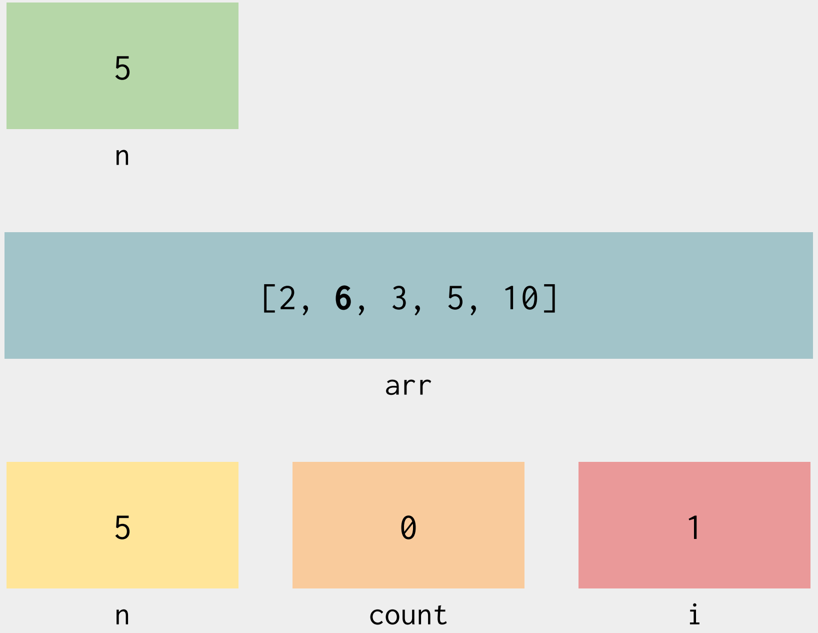
What does this print?

```
// 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;
}
```



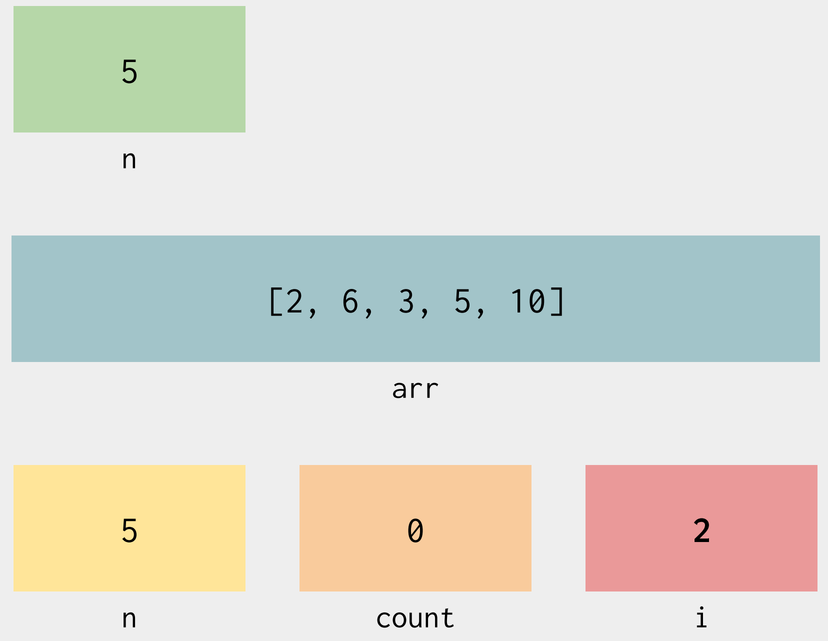
What does this print?

```
// 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;
}
```



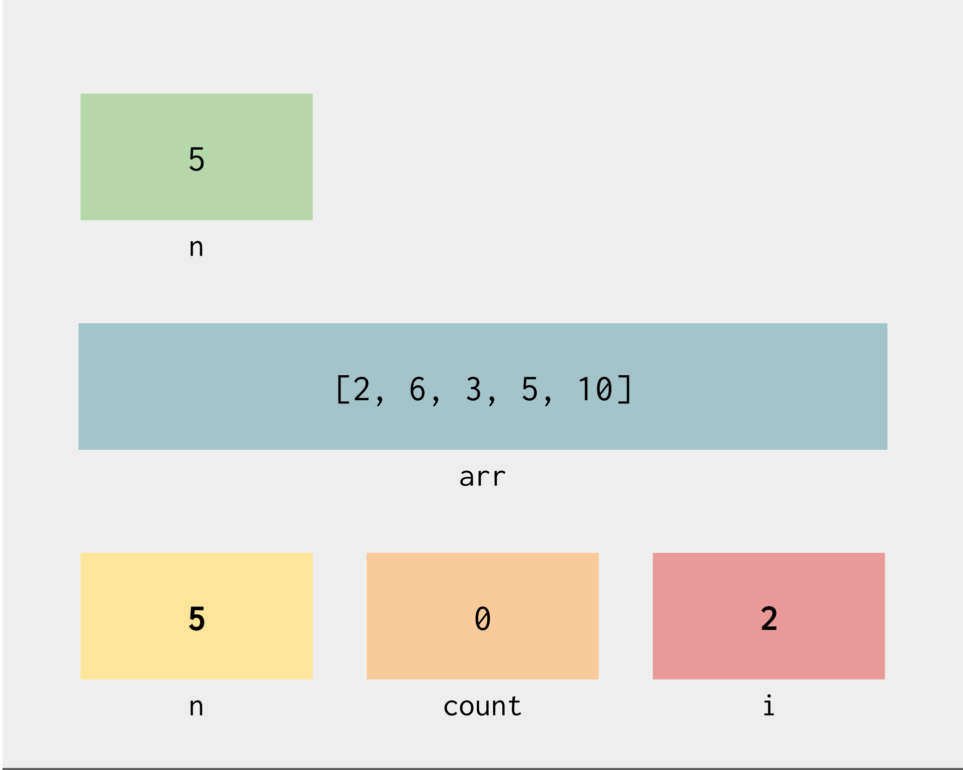
What does this print?

```
// 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;
}
```



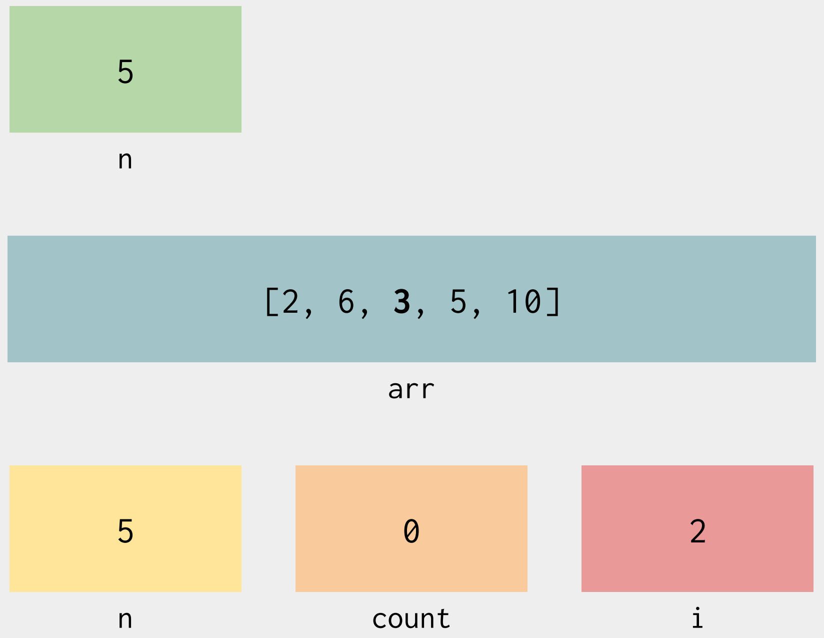
What does this print?

```
// 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;
}
```



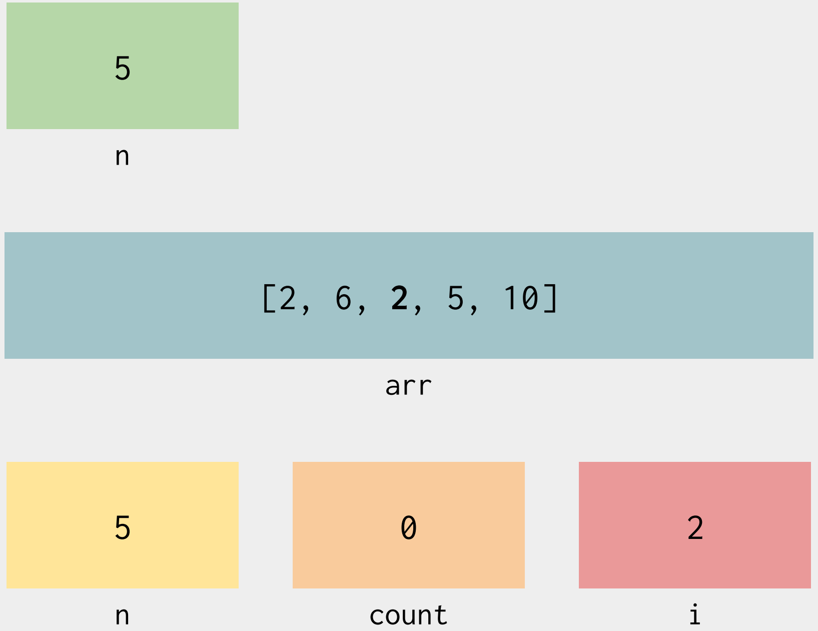
What does this print?

```
// 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;
}
```



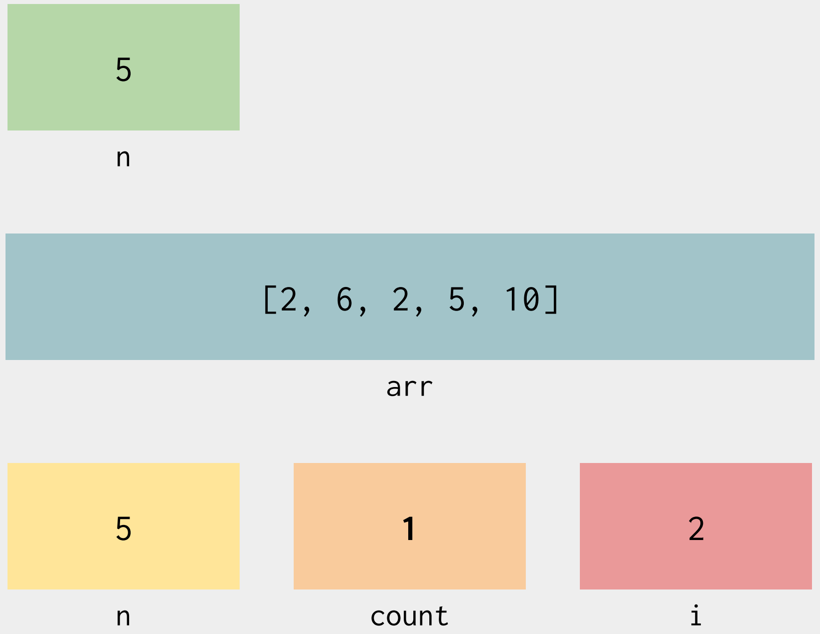
What does this print?

```
// 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;
}
```



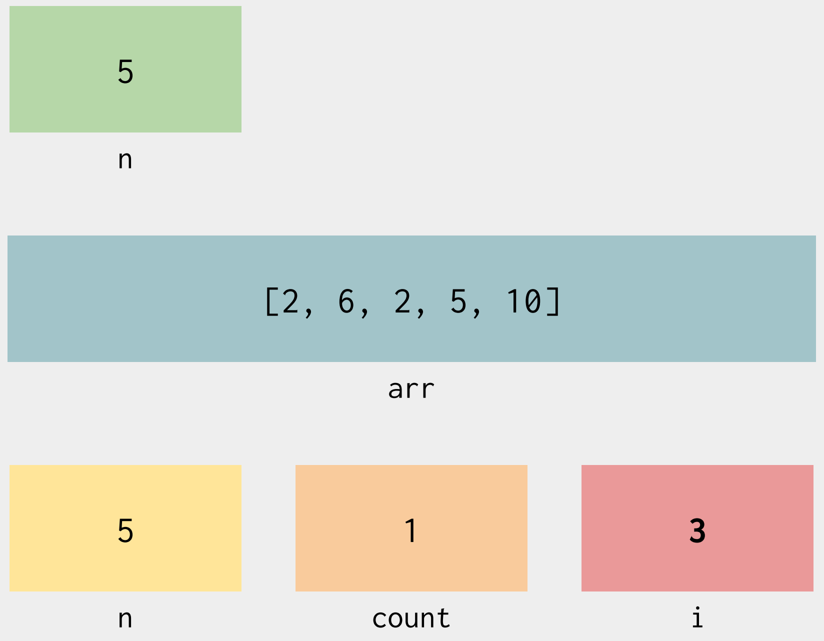
What does this print?

```
// 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;
}
```



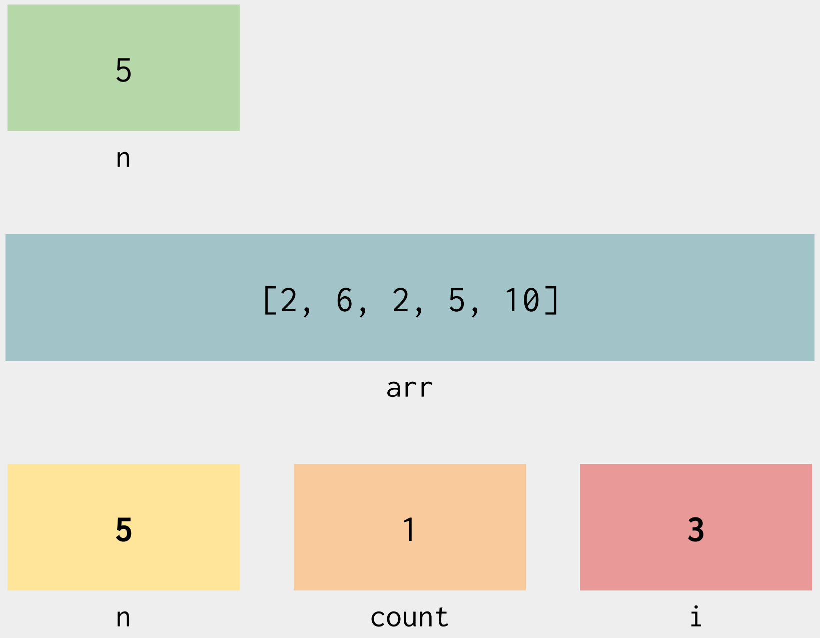
What does this print?

```
// 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;
}
```



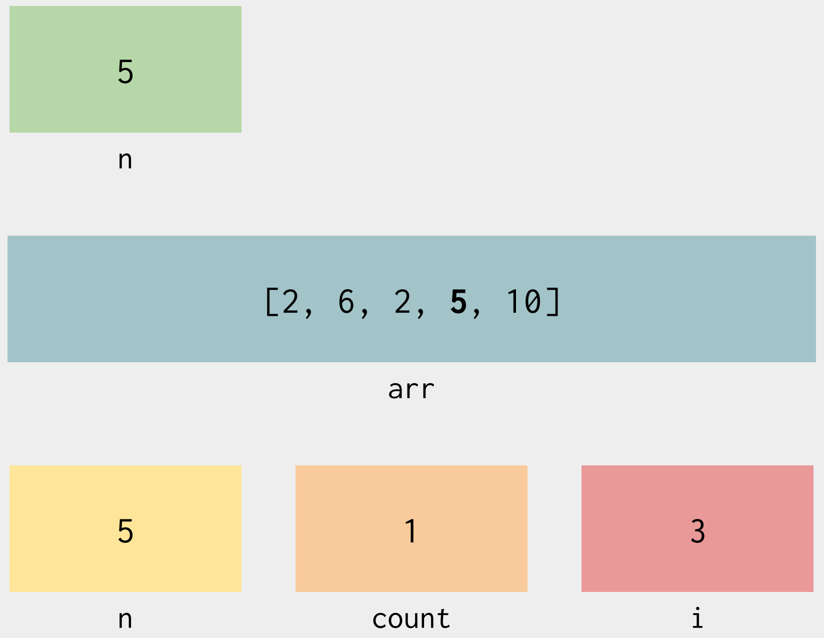
What does this print?

```
// 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;
}
```



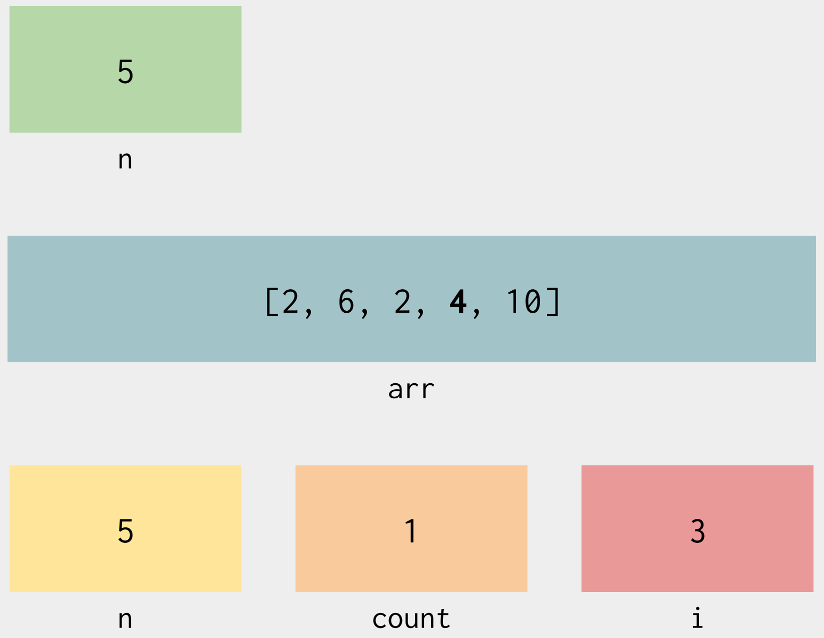
What does this print?

```
// 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;
}
```



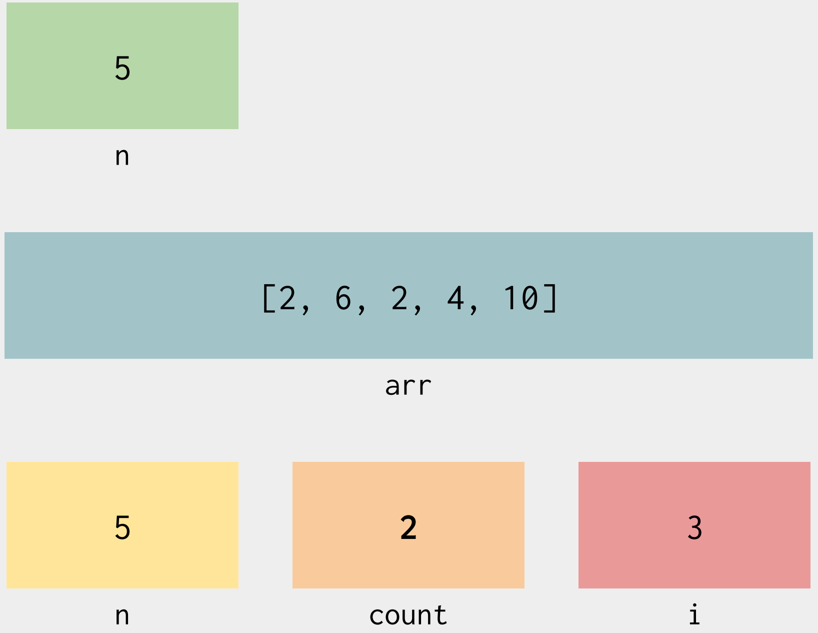
What does this print?

```
// 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;
}
```



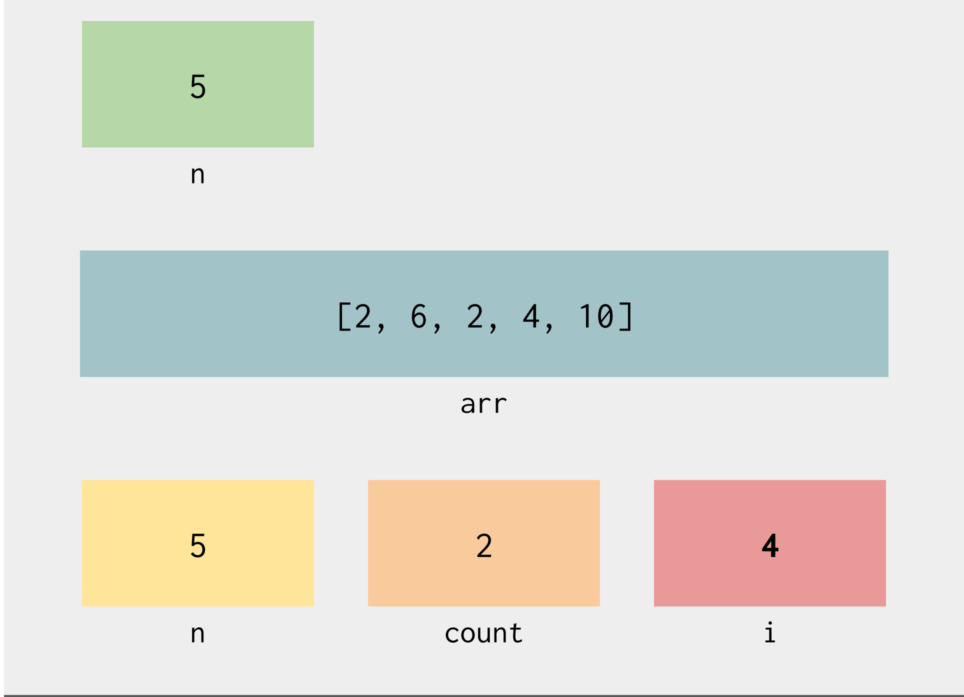
What does this print?

```
// 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;
}
```



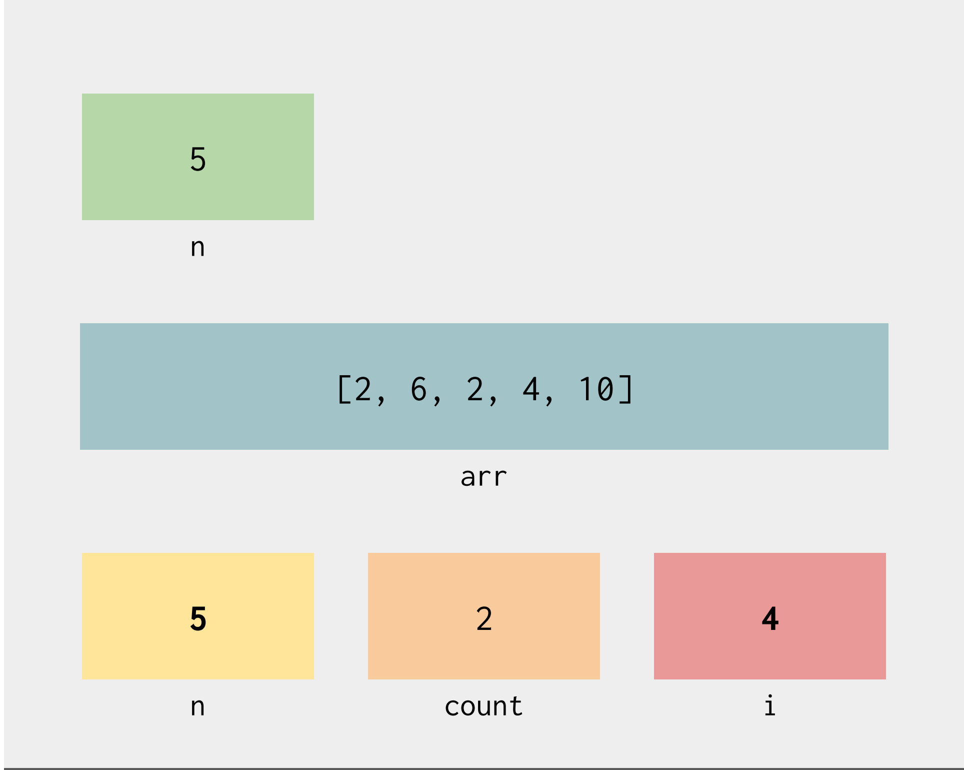
What does this print?

```
// 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;
}
```



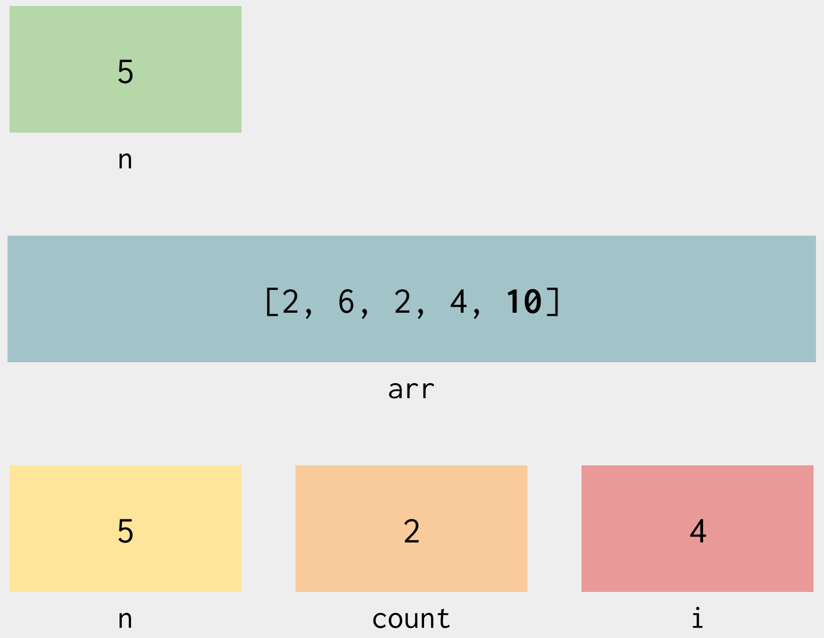
What does this print?

```
// 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;
}
```



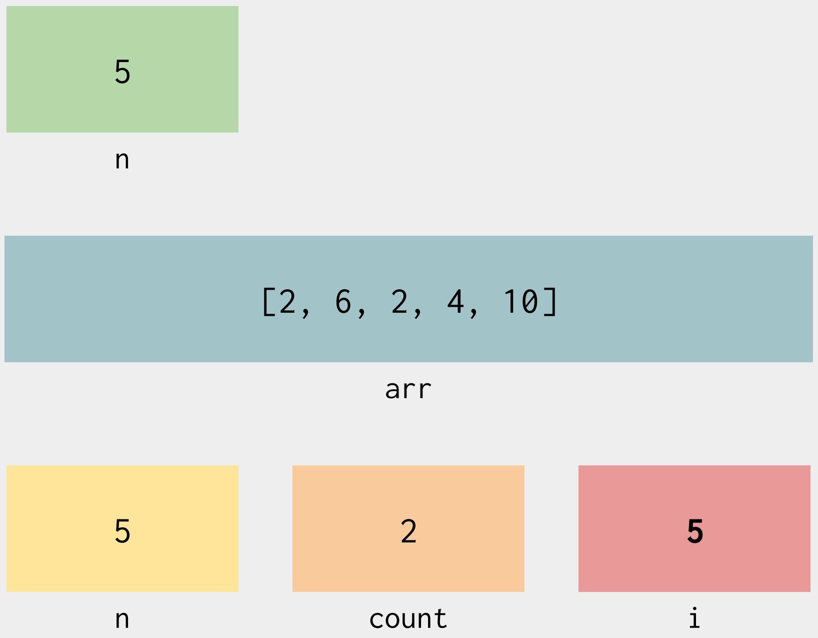
What does this print?

```
// 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;
}
```



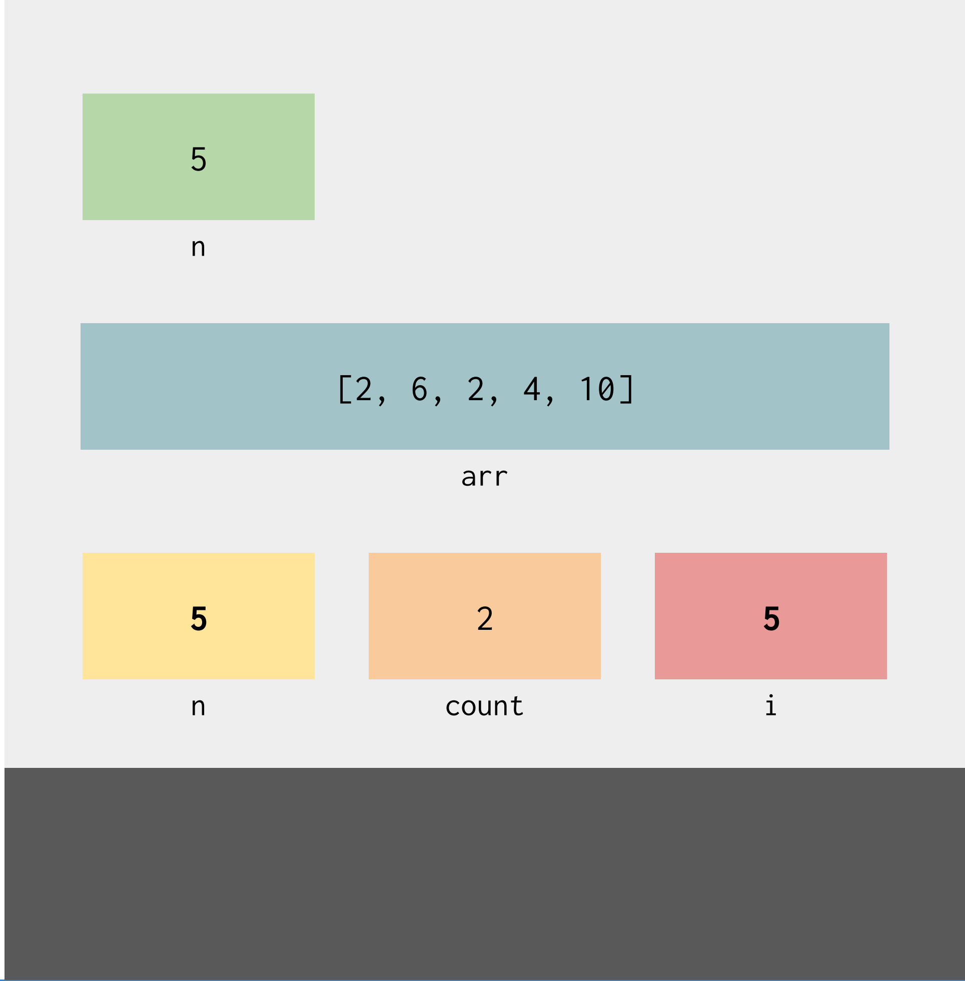
What does this print?

```
// 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;
}
```



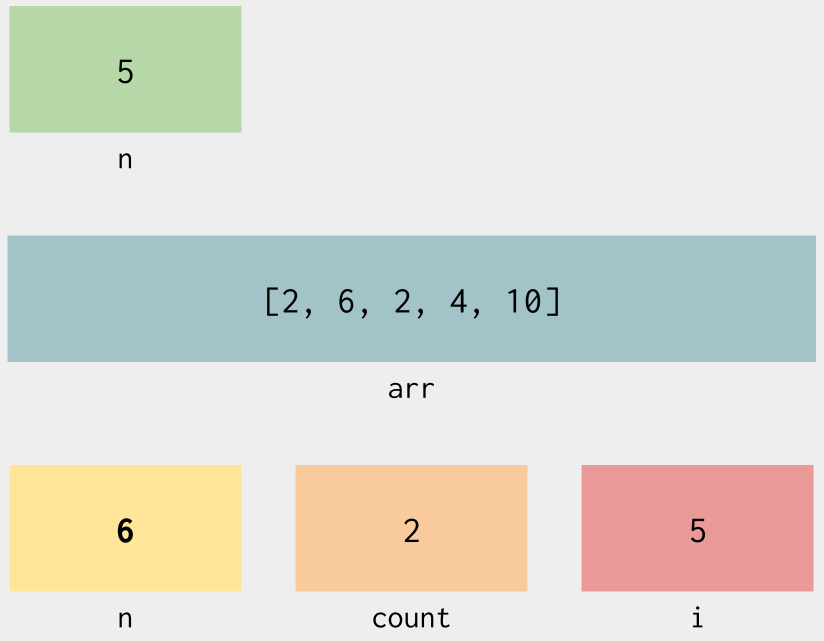
What does this print?

```
// 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;
}
```



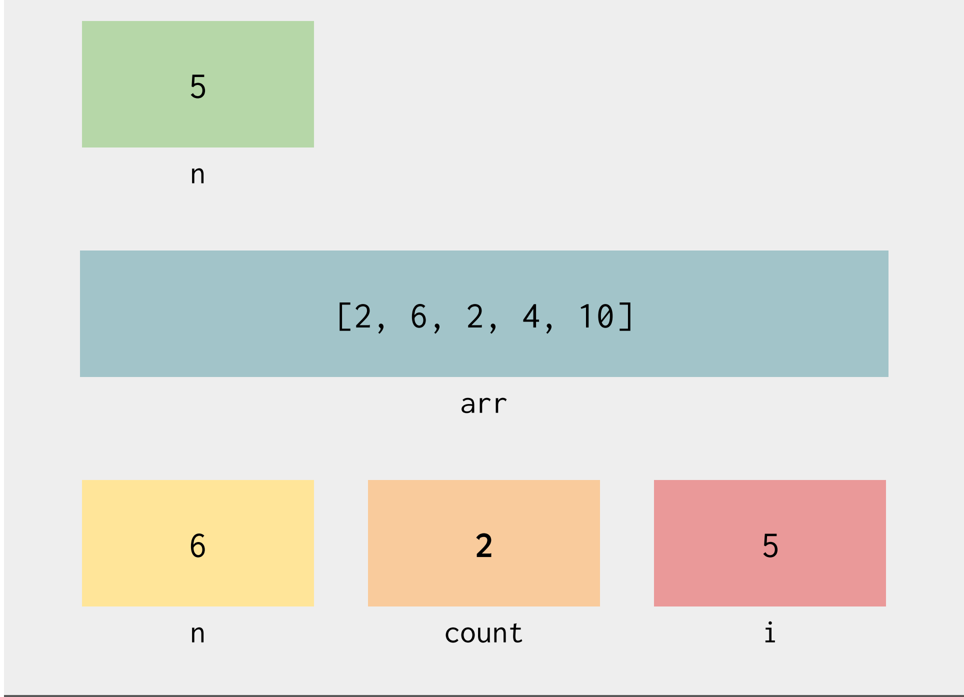
What does this print?

```
// 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;
}
```



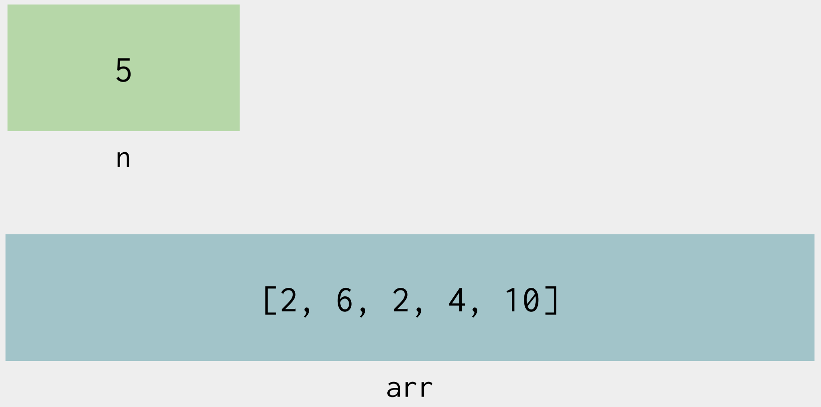
What does this print?

```
// 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;
}
```



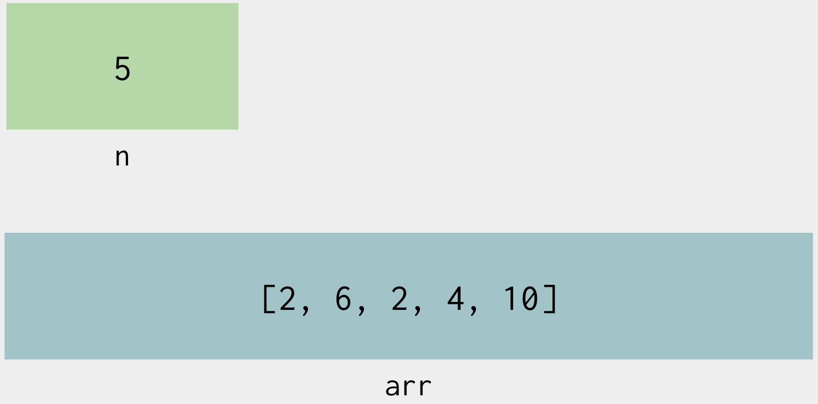
What does this print?

```
// 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;
}
```



```
> 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?

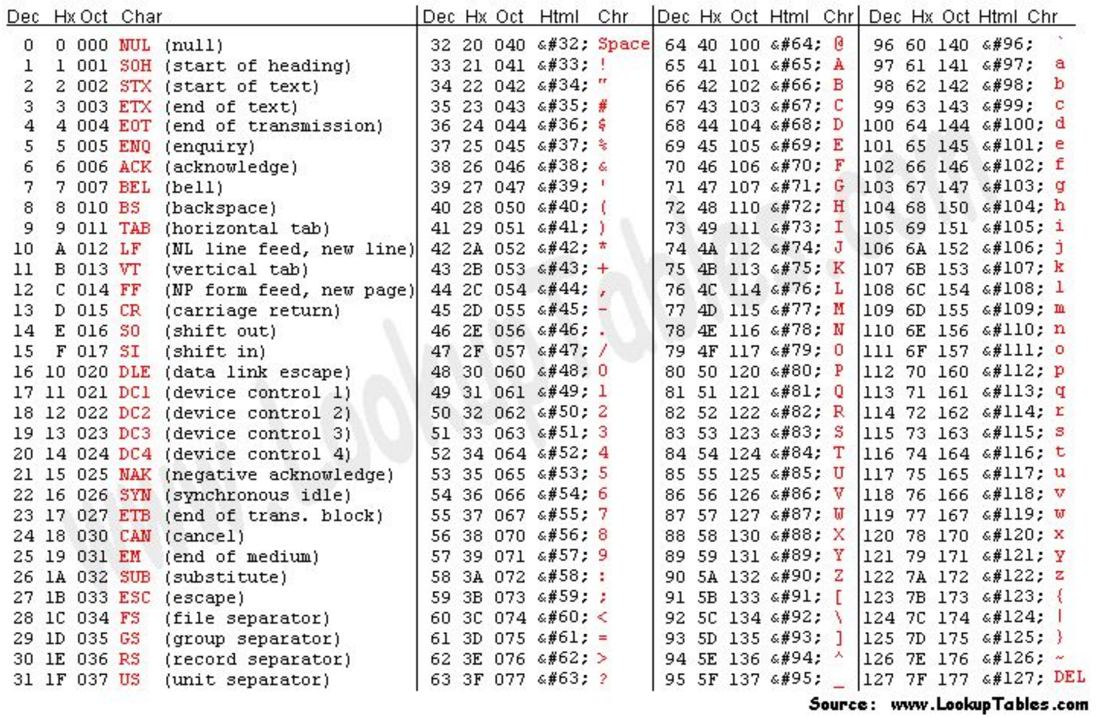
```
// Assume arr only contains n elements
int countFives(int arr[], int n) {
  int count = 0;
  for (int i = 0; i <= n; ++i) {
    if (arr[i] == 5) {
      count++;
    }
  }
  return count;
}
```

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

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

Source: www.LookupTables.com

Ascii (cont.)

```
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.
```

C Strings (cont.)

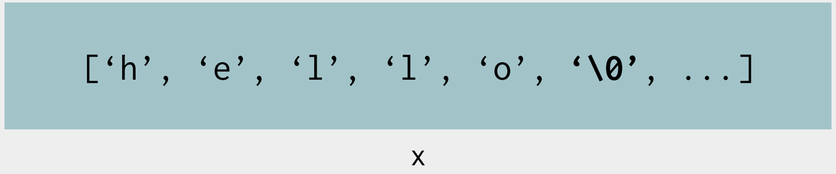
- The end of a C string is marked by a null byte (`‘\0’`)
    - Null byte has ASCII value `0`
    - **`strlen`** simply looks for the null byte for you

```
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.)

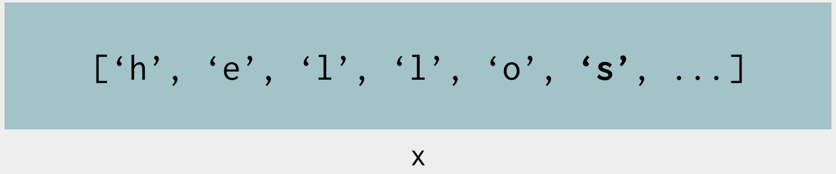
```
// A null character is automatically
// put in index 5.
char x[50] = “hello”;

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



```
[‘h’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’, ...]
```

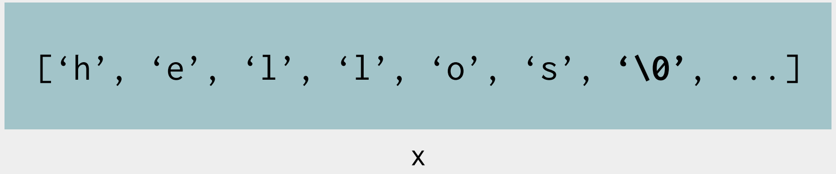
C Strings (cont.)

```
// A null character 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 null character 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’;
```



Pointers

- A **pointer** is the memory address of a variable.

- The **&** operator can be used to determine the **address** of a variable to be stored in the pointer.
- The ***** operator can be used to **dereference** a pointer and get the value stored in the variable that is being pointed to.

```
double d = 10.0;
double *dp;          // pointer that holds the address of a double
dp = &d;             // stores the address of d into dp
*dp = 20.0;          // changes the value of d from 10.0 to 20.0
```

Pointers

```
int var = 20;    // actual variable declaration
int *ip;         // pointer variable declaration

// store address of var in ip
ip = &var;

cout << "Value of var variable: ";
cout << var << endl;

// print the address stored in ip pointer
cout << "Address stored in ip variable: ";
cout << ip << endl;

// access the value at address stored in pointer
cout << "Value of *ip variable: ";
cout << *ip << endl;
```

```
> Value of var variable: 20
> Address stored in ip variable: 0xBFC601AC
> Value of *ip variable: 20
```

Pointer Arithmetic

```
int arr[] = {10, 20, 30, 40};

int *ptr = arr;               // arr == &arr[0]
cout << *ptr << endl;

ptr++;
cout << *ptr << endl;

ptr = ptr + 1;
cout << *ptr << endl;

ptr--;
cout << *(ptr + 2) << endl;
```

```
> 10
> 20
> 30
> 40
```

Pointers – new and delete

- The **`new`** operator can be used to create **dynamic** variables. These variables can be accessed using pointers.

```
string *p;
p = new string;
p = new string(“hello”);
```

- The **`delete`** operator eliminates dynamic variables.

```
delete p;
```

- Note: Pointer p is now a **dangling pointer**! Dereferencing it is dangerous and leads to undefined behavior. One way to avoid this is to set p to `NULL` after using `delete`.

Pointers – Dynamic Arrays

- A pointer can also be used when creating a *dynamic* array. Dynamic arrays are useful because their size can be determined while the program is running!

```
int *ptr;
int arraySize;
cin >> arraySize;
ptr = new int[arraySize];
```

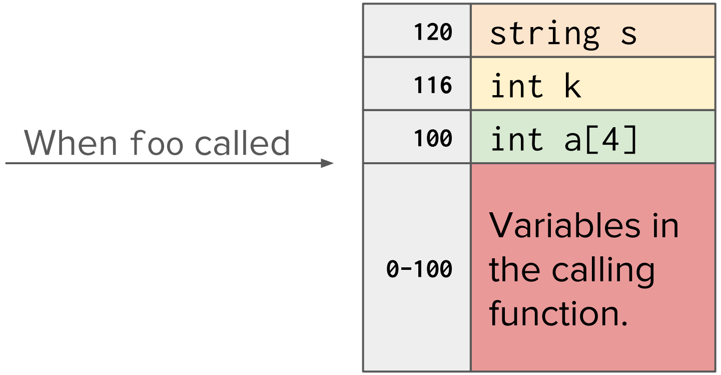
- To destroy the dynamically allocated array, use the **`delete[]`** operator.

```
delete[] ptr;
```

Pointers – the Heap and the Stack

- As it turns out, there are **two** places where your variables live.
- The first is the **stack**, which is the place you're most familiar with. With **local variables**, the compiler is like a city planner who decides where each variable should live.

```
void foo() {
  int a[4];  // Stored at 100
  int k;     // Stored at 116
  string s;  // Stored at 120
}
```

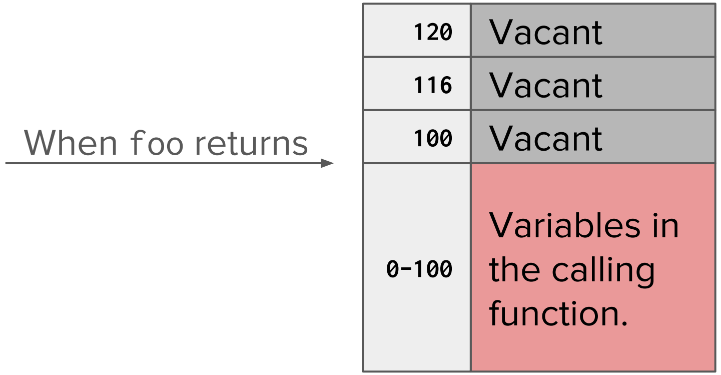


If the size isn't specified at compile time, how would the compiler know where to put k or s?

Pointers – the Heap and the Stack (cont.)

- As it turns out, there are **two** places where your variables live.
- The first is the **stack**, which is the place you're most familiar with. With **local variables**, the compiler is like a city planner who decides where each variable should live.

```
void foo() {
  int a[4];  // Stored at 100
  int k;     // Stored at 116
  string s;  // Stored at 120
}
```

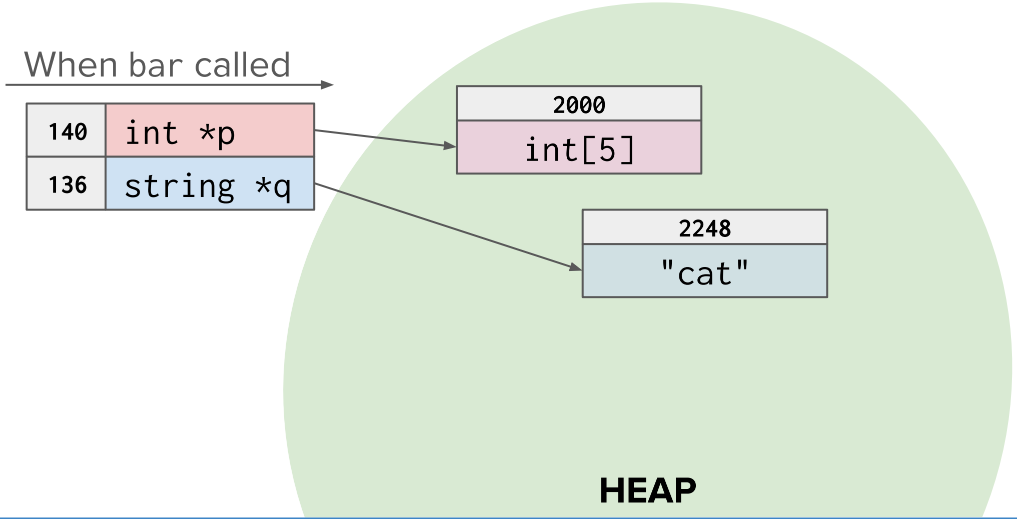


When the function returns, the variables are evicted from their addresses.

Pointers – the Heap and the Stack (cont.)

- As it turns out, there are **two** places where your variables live.
- The second is the **heap**, which is the place where dynamic variables live. Dynamic variables essentially lease some part of the heap to live in.

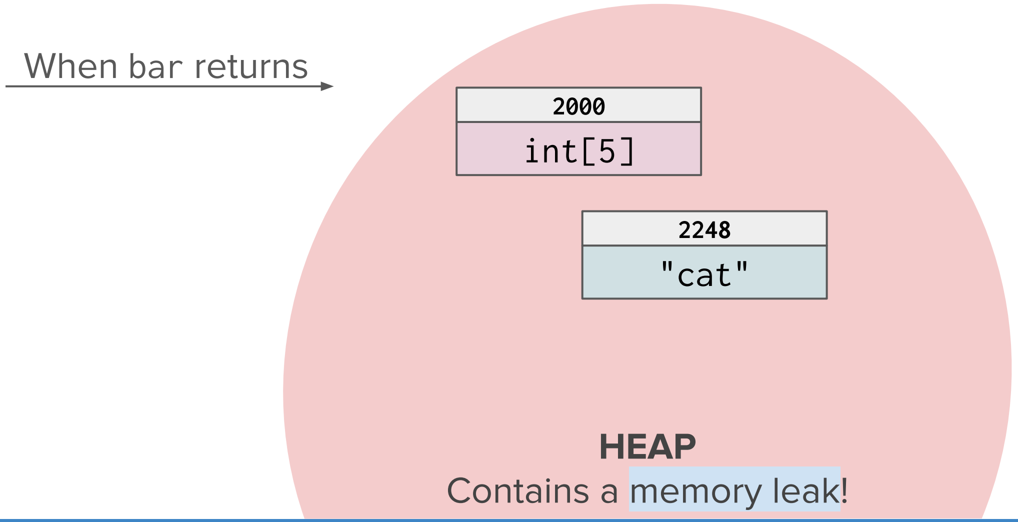
```
void bar() {
  int *p = new int[5];
  string *q = new string("Cat");
}
```



Pointers – the Heap and the Stack (cont.)

- As it turns out, there are **two** places where your variables live.
- The second is the **heap**, which is the place where dynamic variables live. Dynamic variables essentially lease some part of the heap to live in.

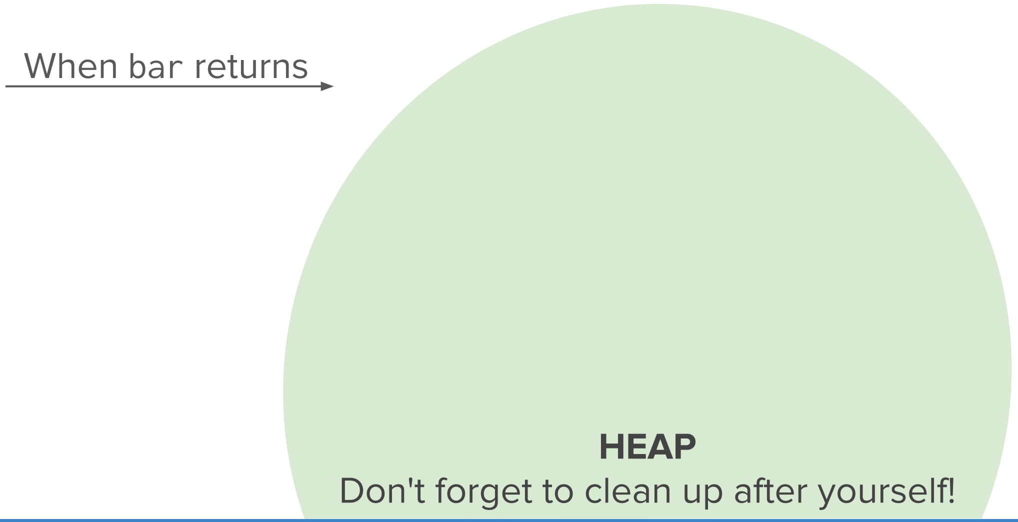
```
void bar() {
  int *p = new int[5];
  string *q = new string("Cat");
}
```



Pointers – the Heap and the Stack (cont.)

- As it turns out, there are **two** places where your variables live.
- The second is the **heap**, which is the place where dynamic variables live. Dynamic variables essentially lease some part of the heap to live in.

```
void bar() {
  int *p = new int[5];
  string *q = new string("Cat");
  delete[] p;
  delete q;
}
```



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 repeated element. You may assume that there will be at least one duplicate element in the array.

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 two for loops to check every character. Once we find a repeated character, we update the index only if it is less than minIndex, which is initiated to the value n - 1 which is the largest possible value.

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

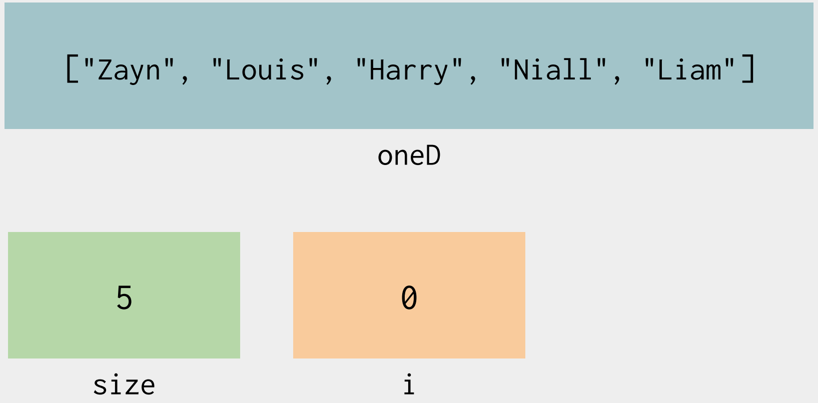
Practice Question: What Makes CS Beautiful

```
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?

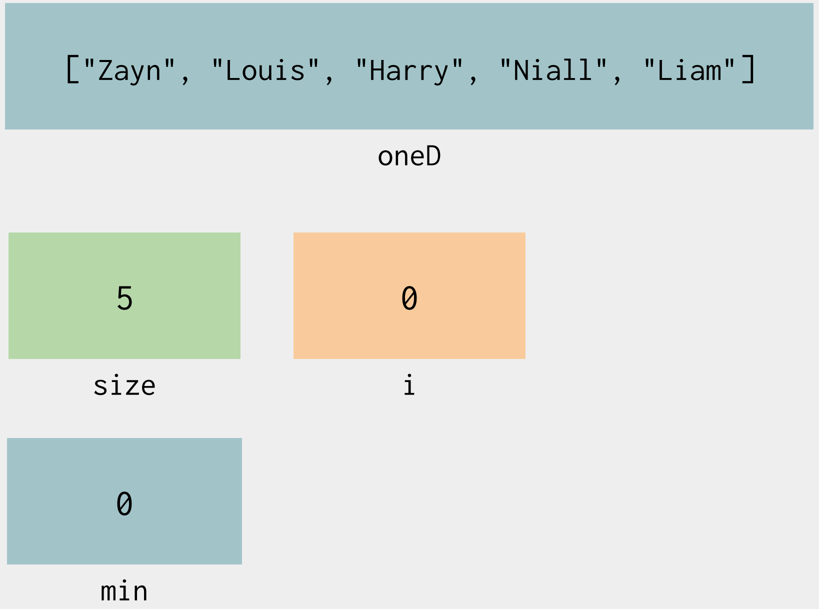
Walkthrough

```
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];
```



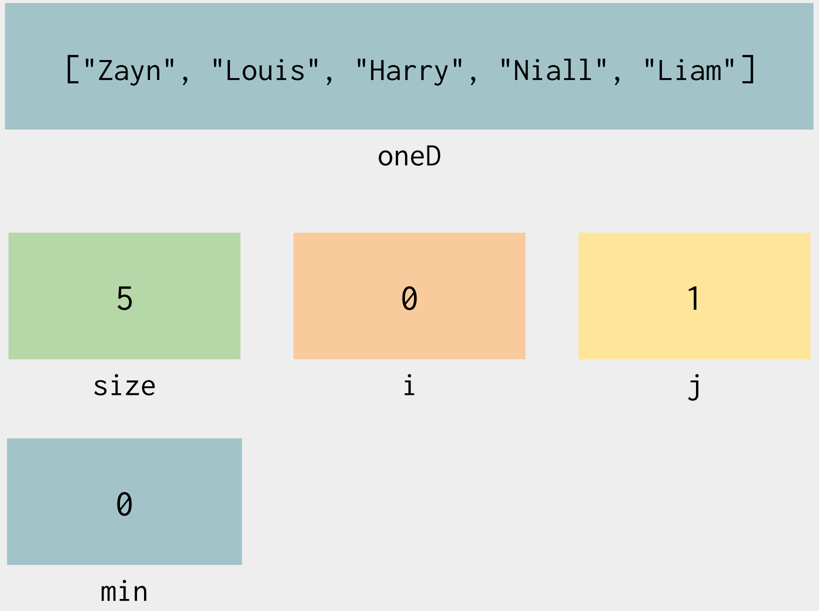
Walkthrough

```
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];
```



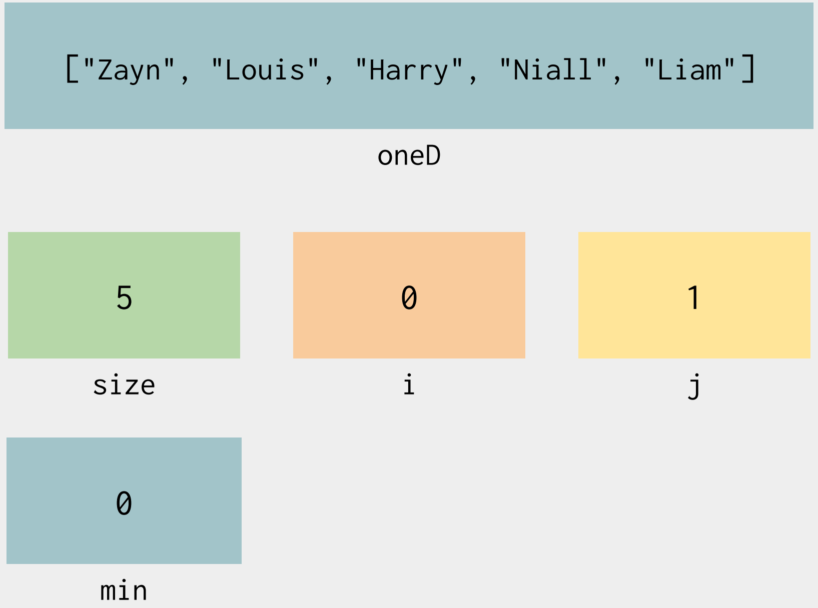
Walkthrough

```
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];
```



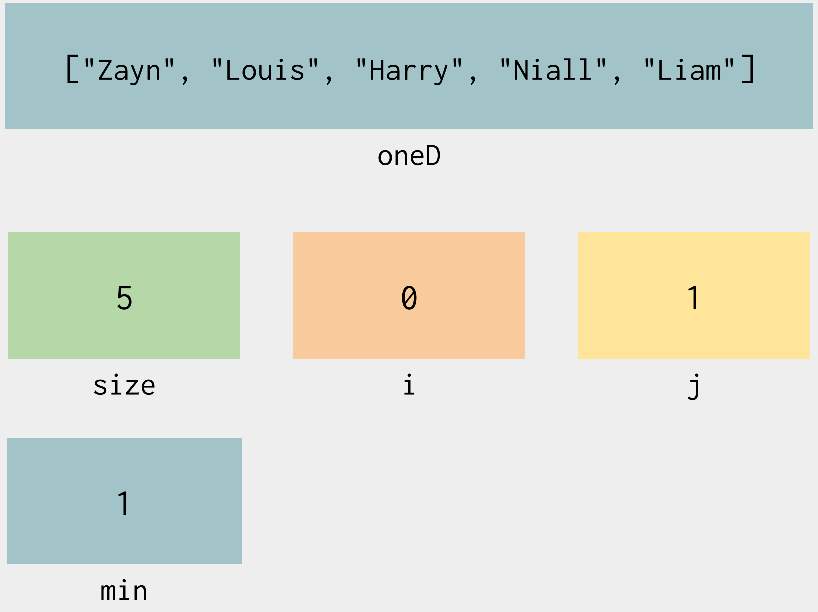
Walkthrough

```
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];
```



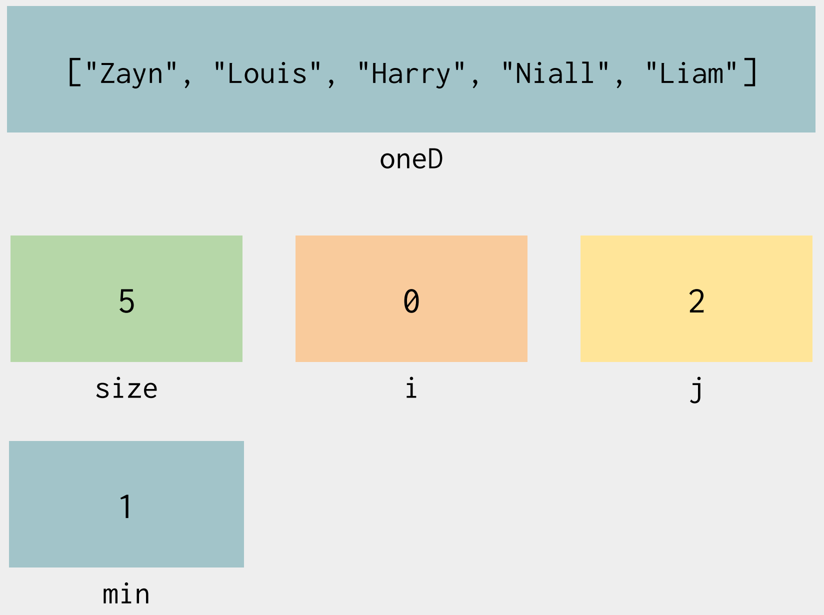
Walkthrough

```
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];
```



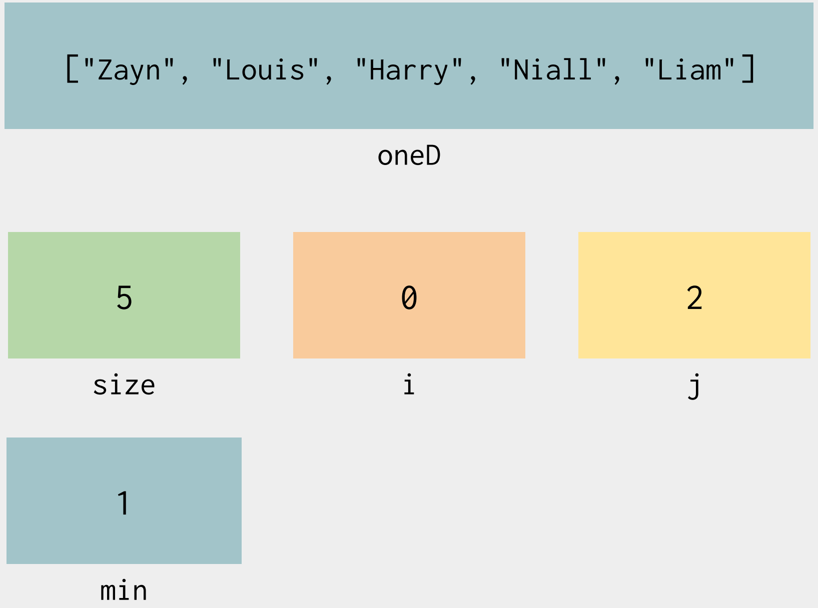
Walkthrough

```
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];
```



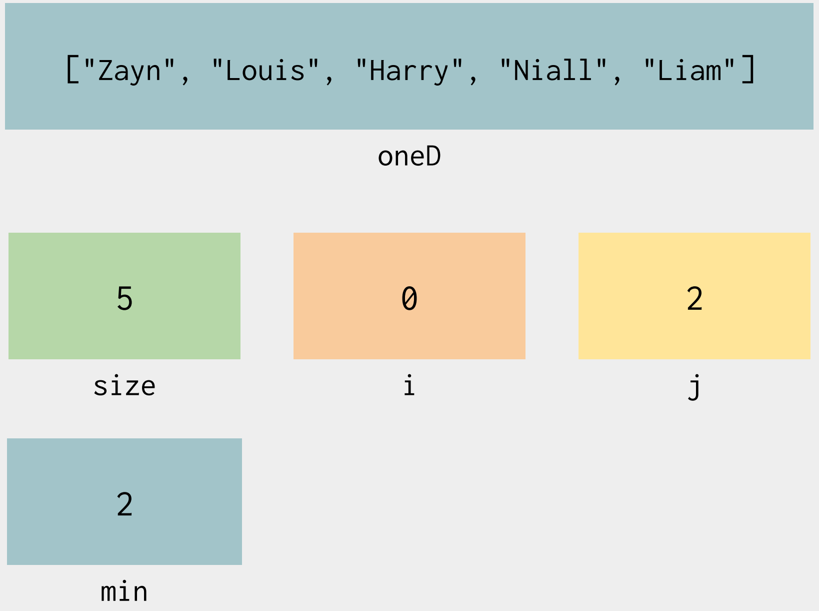
Walkthrough

```
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];
```



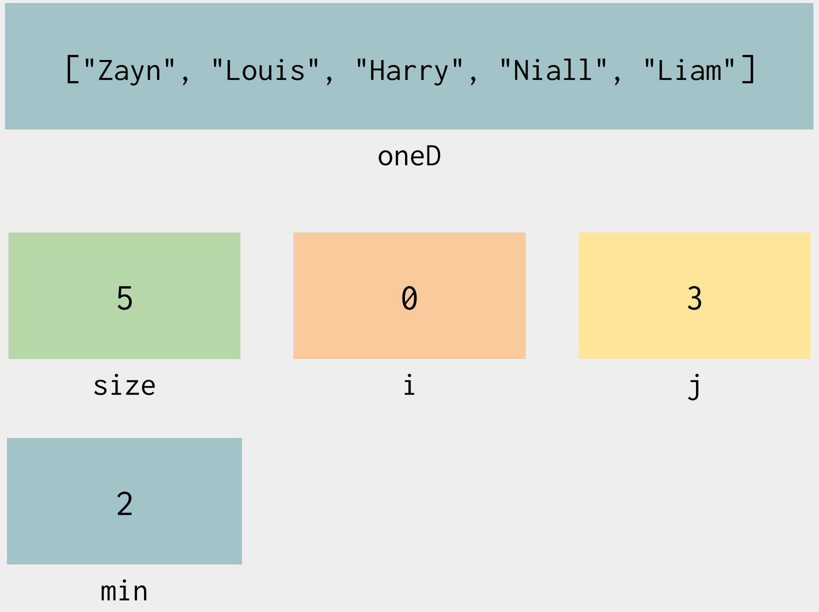
Walkthrough

```
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];
```



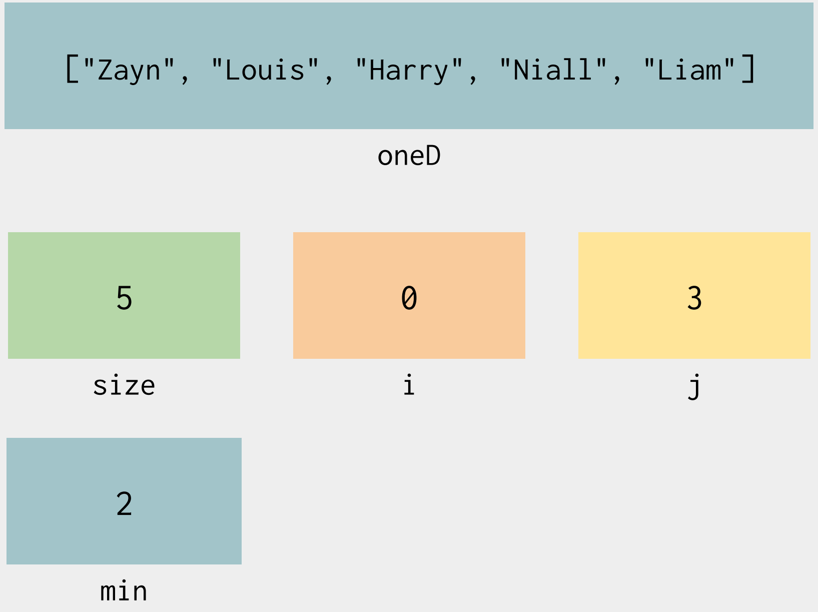
Walkthrough

```
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];
```



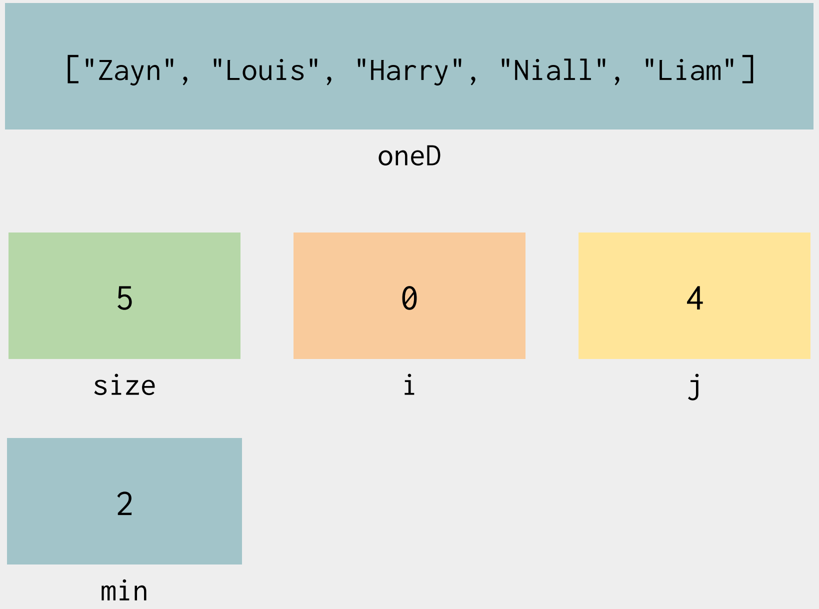
Walkthrough

```
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];
```



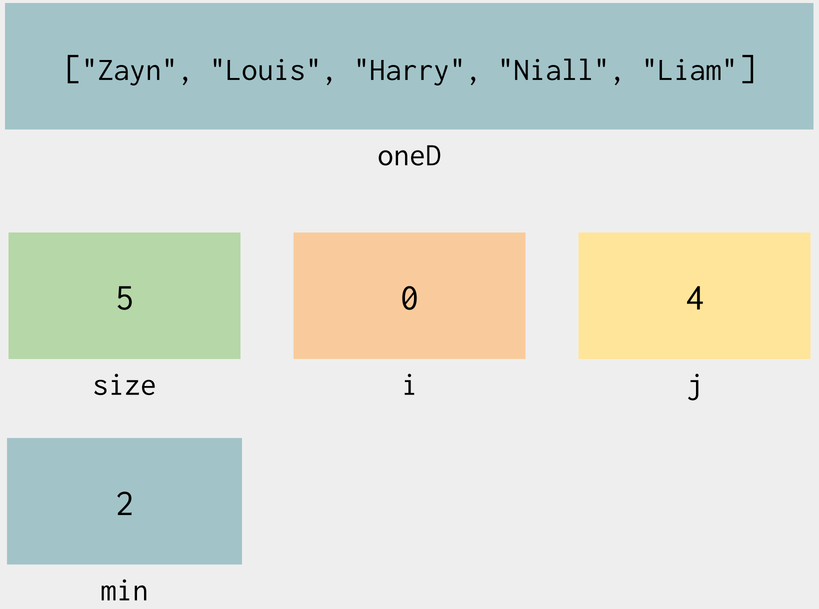
Walkthrough

```
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];
```



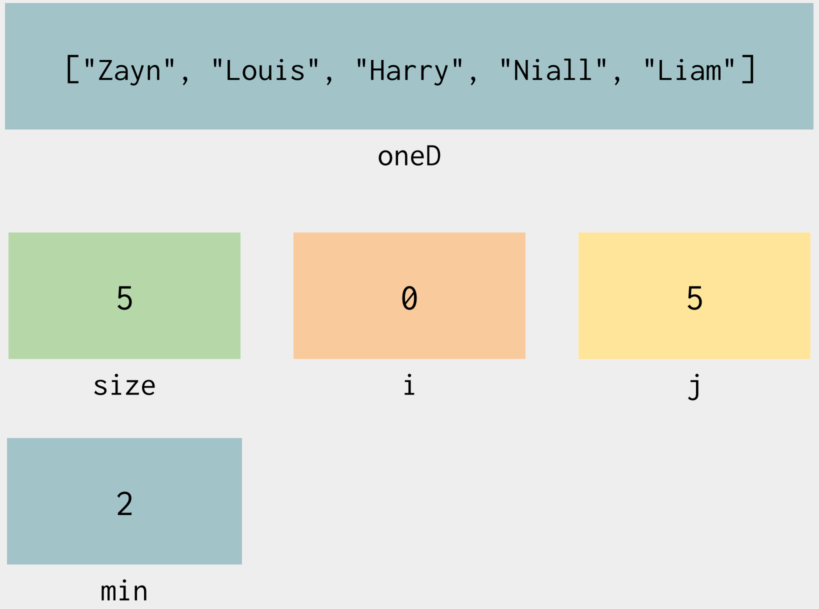
Walkthrough

```
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];
```



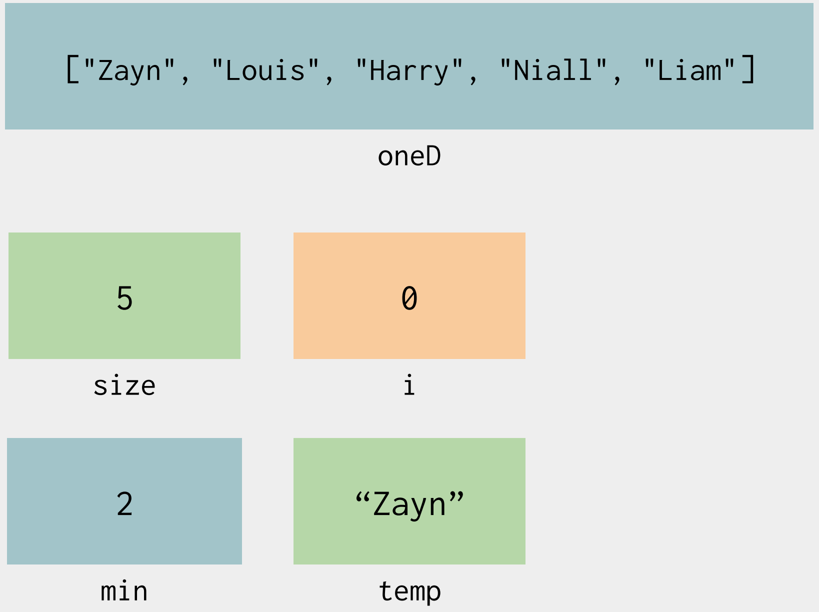
Walkthrough

```
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];
```



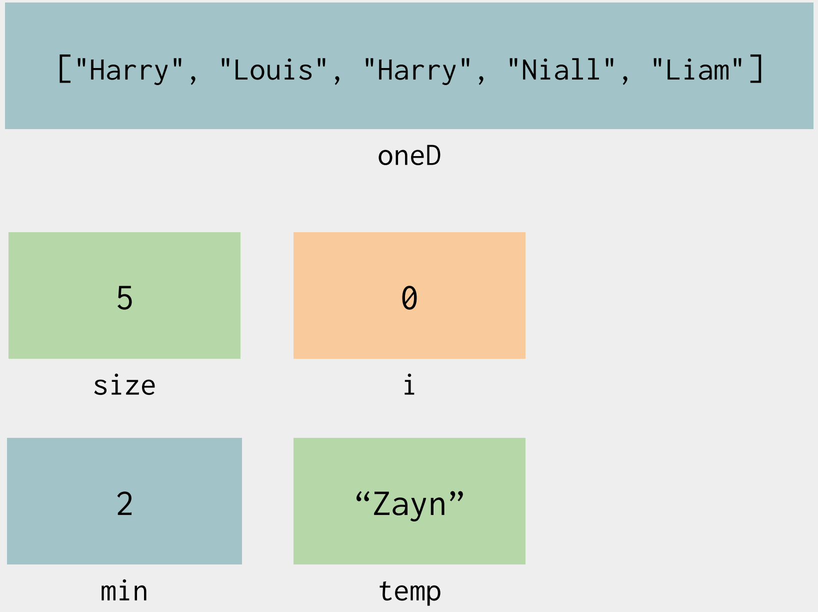
Walkthrough

```
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];
```



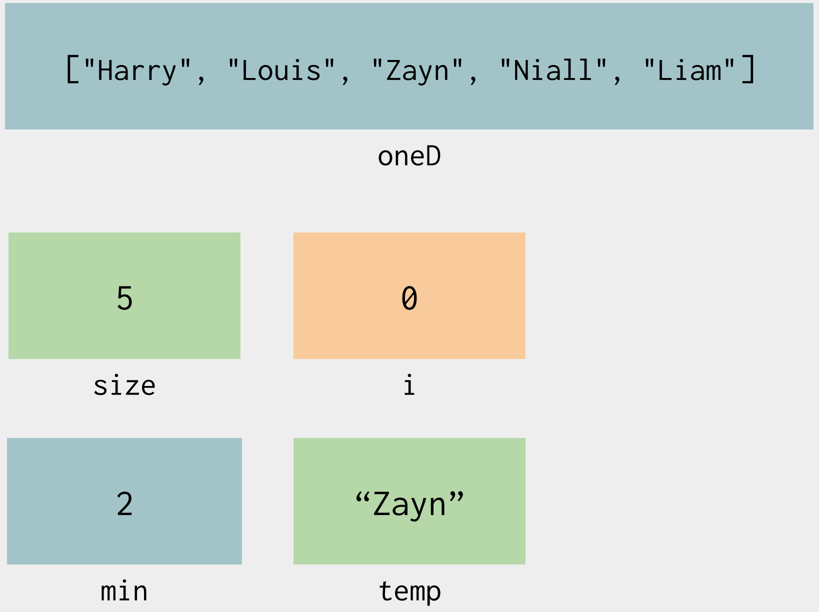
Walkthrough

```
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];
```



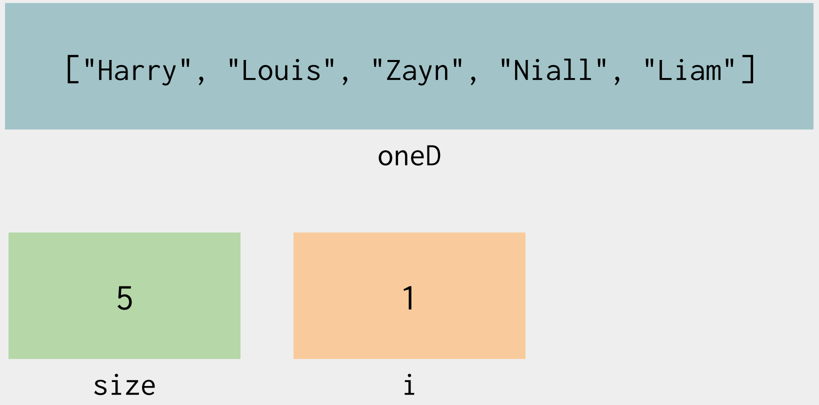
Walkthrough

```
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];
```



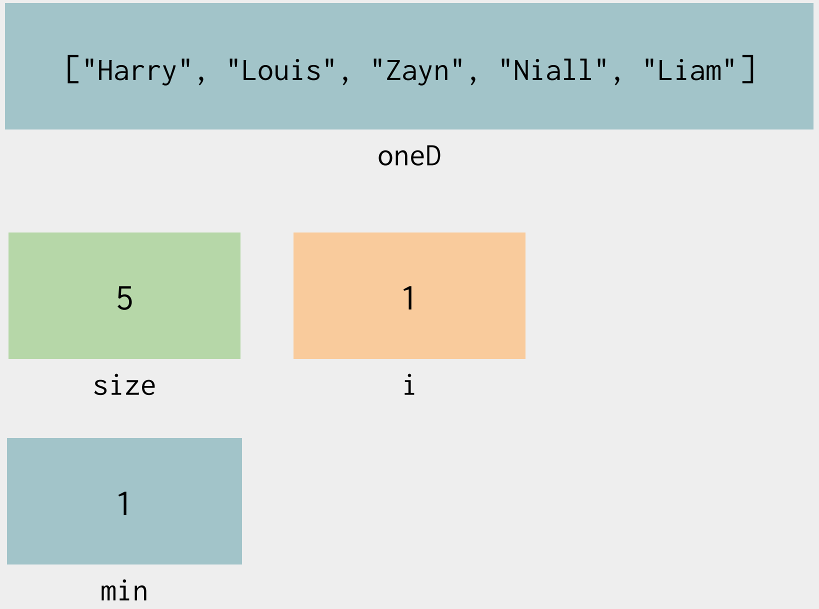
Walkthrough

```
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];
```



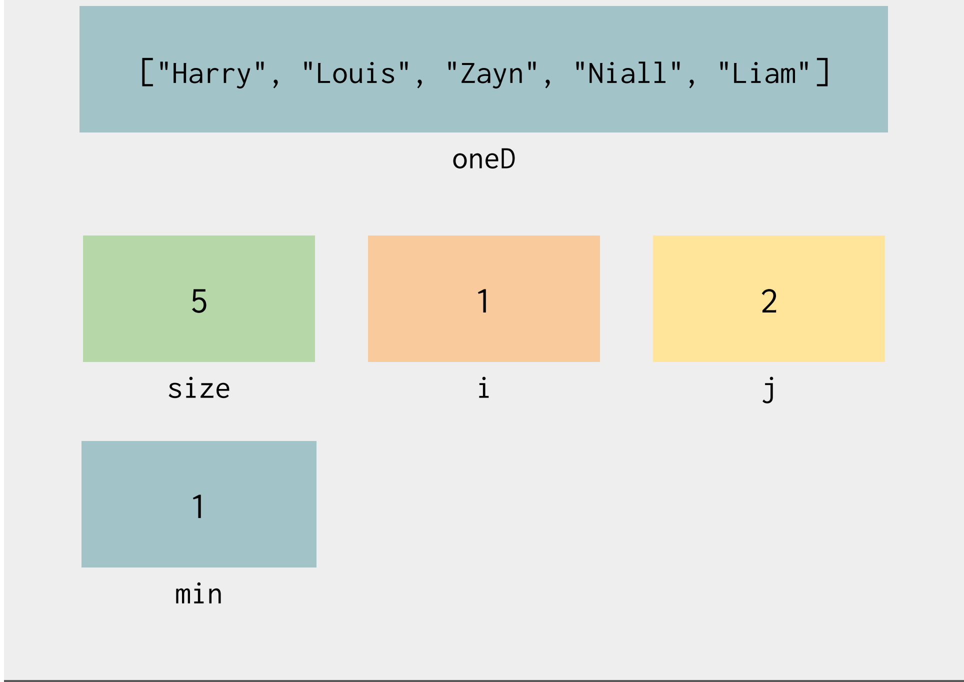
Walkthrough

```
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];
```



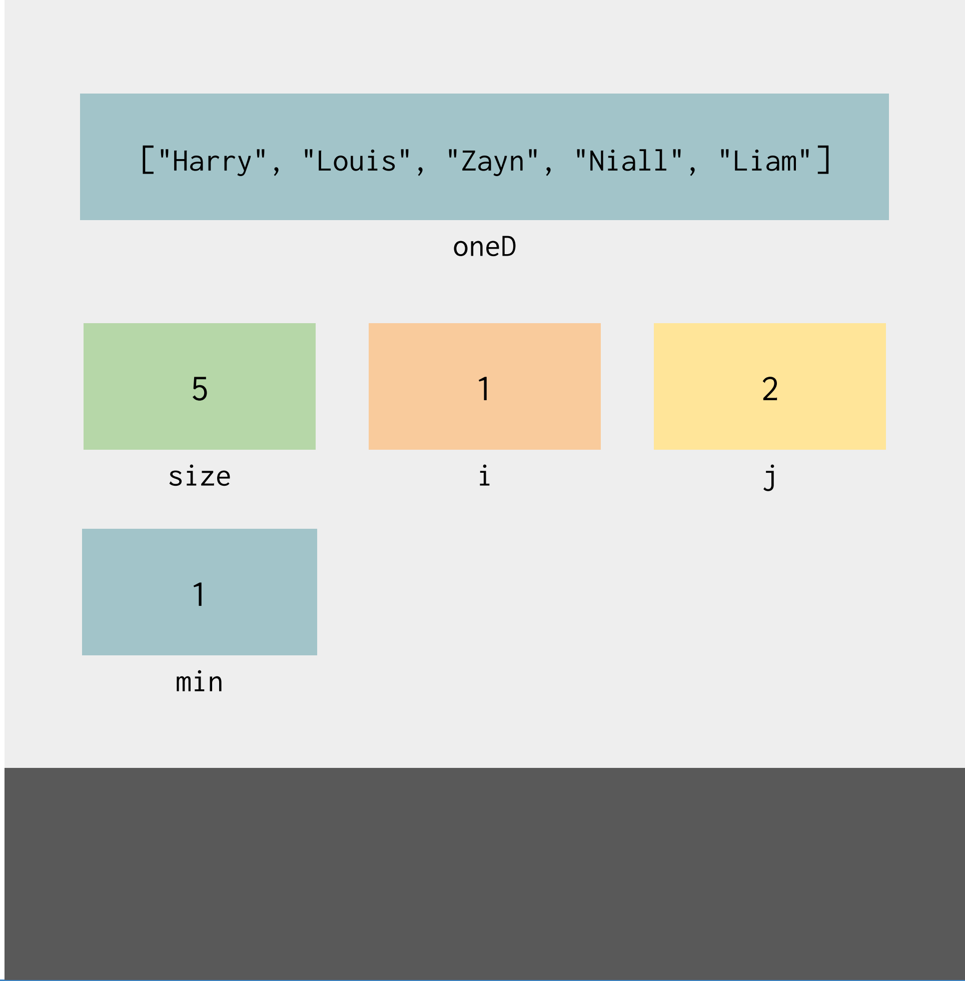
Walkthrough

```
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];
```



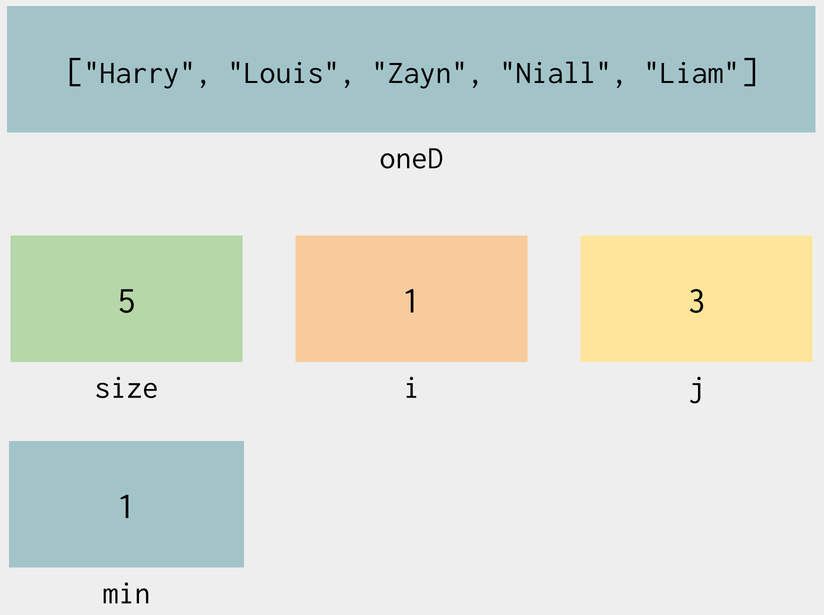
Walkthrough

```
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];
```



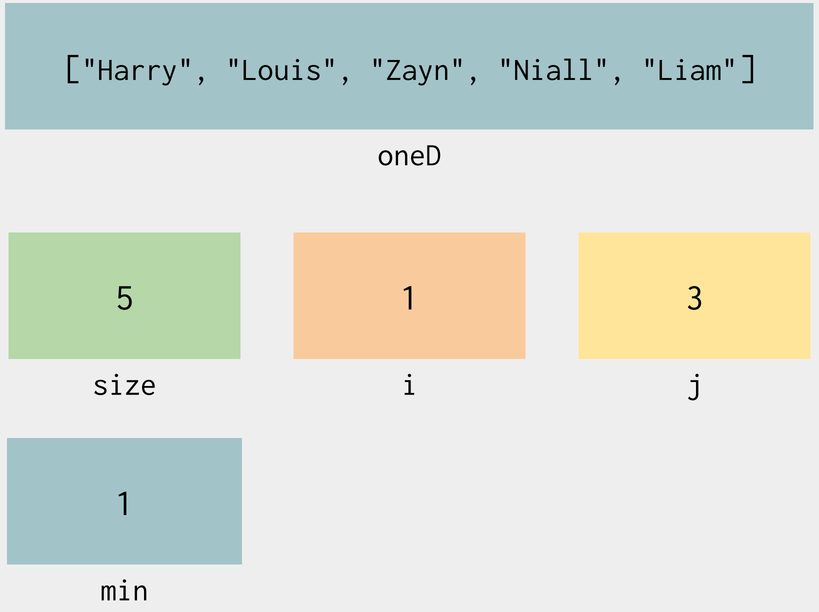
Walkthrough

```
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];
```



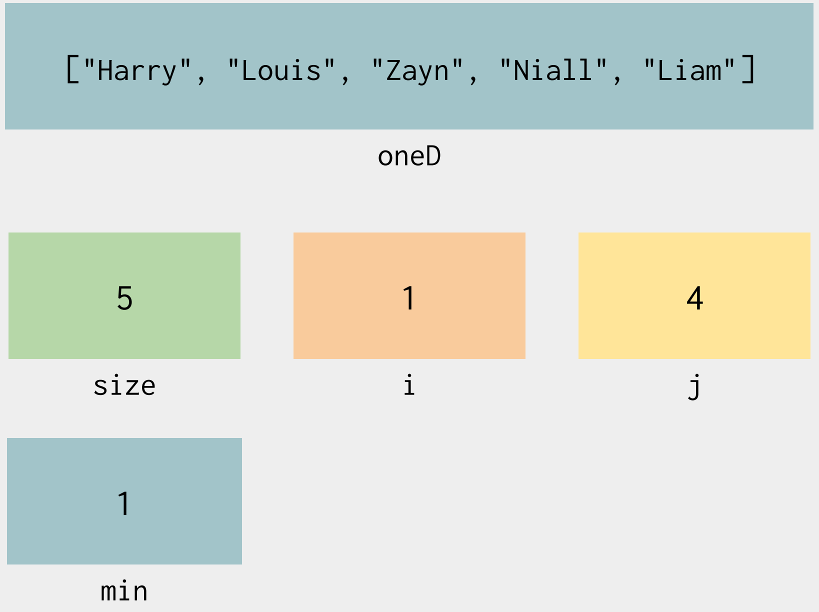
Walkthrough

```
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];
```



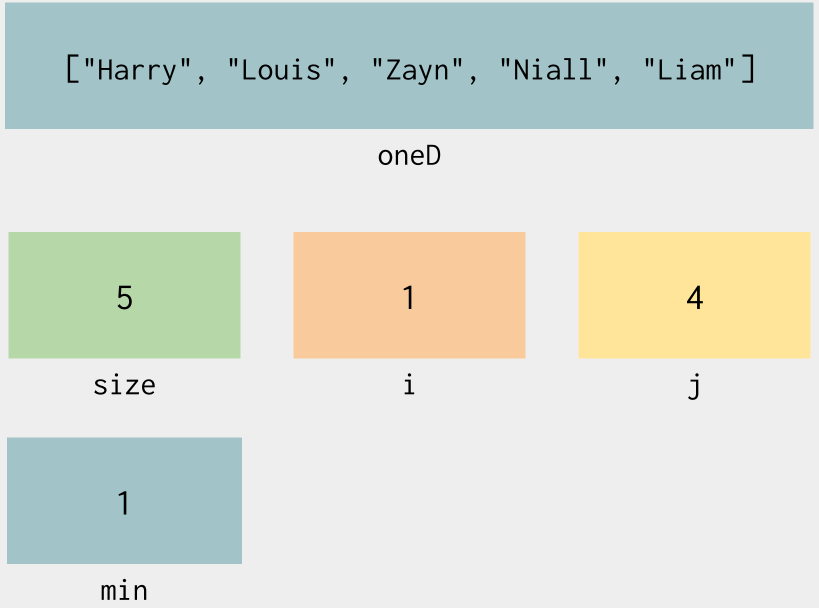
Walkthrough

```
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];
```



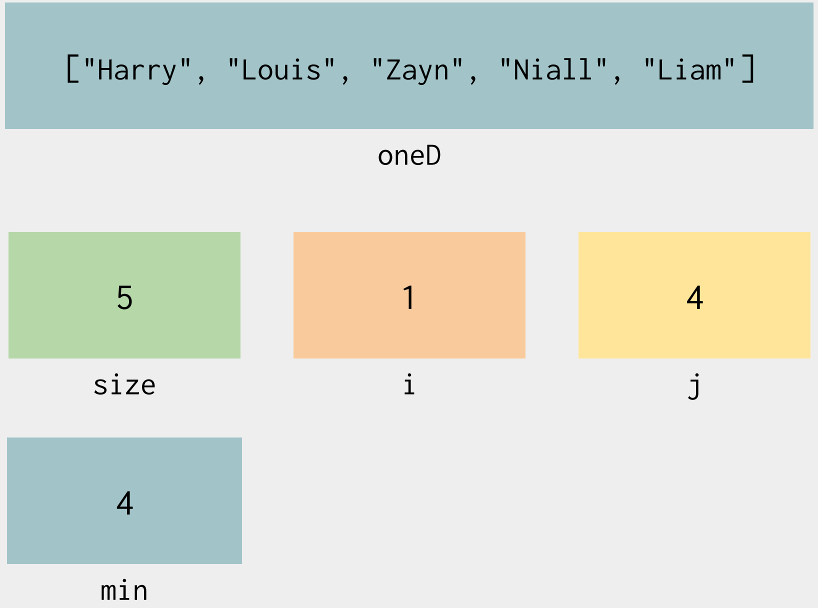
Walkthrough

```
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];
```



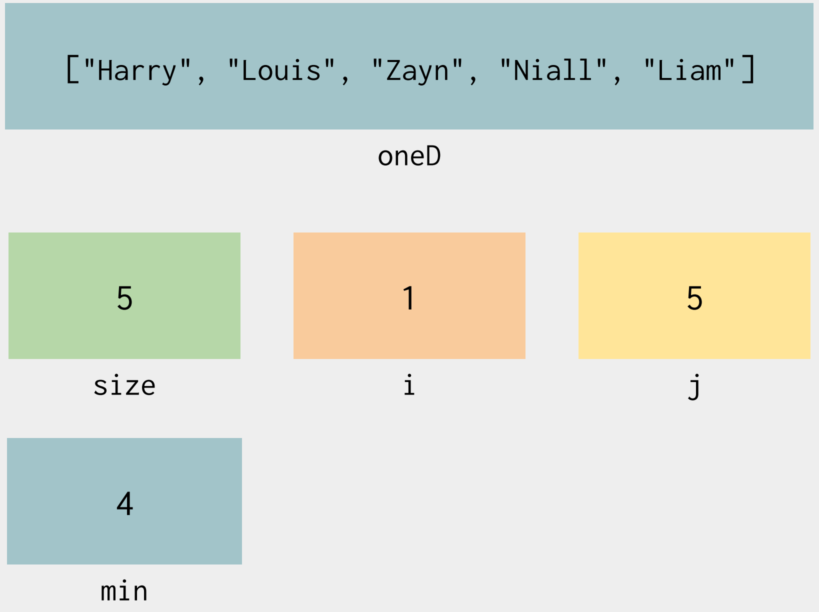
Walkthrough

```
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];
```



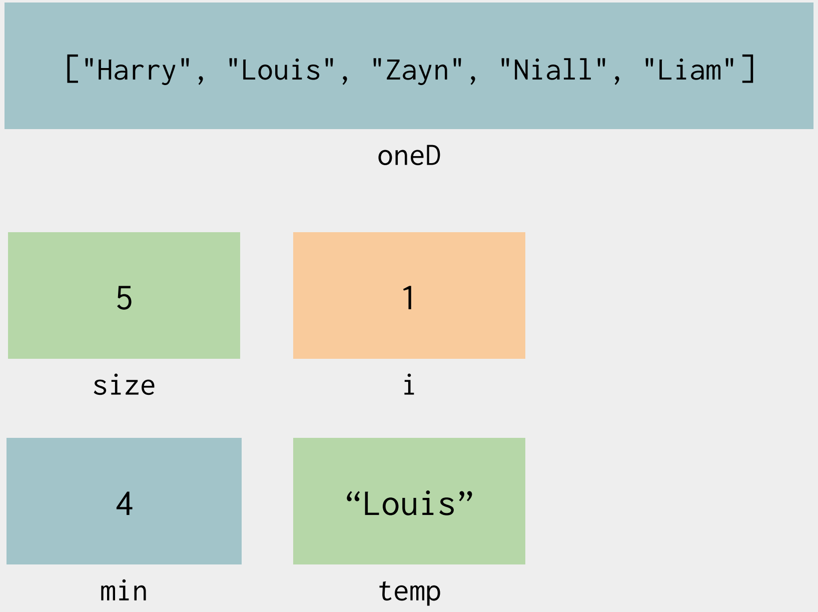
Walkthrough

```
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];
```



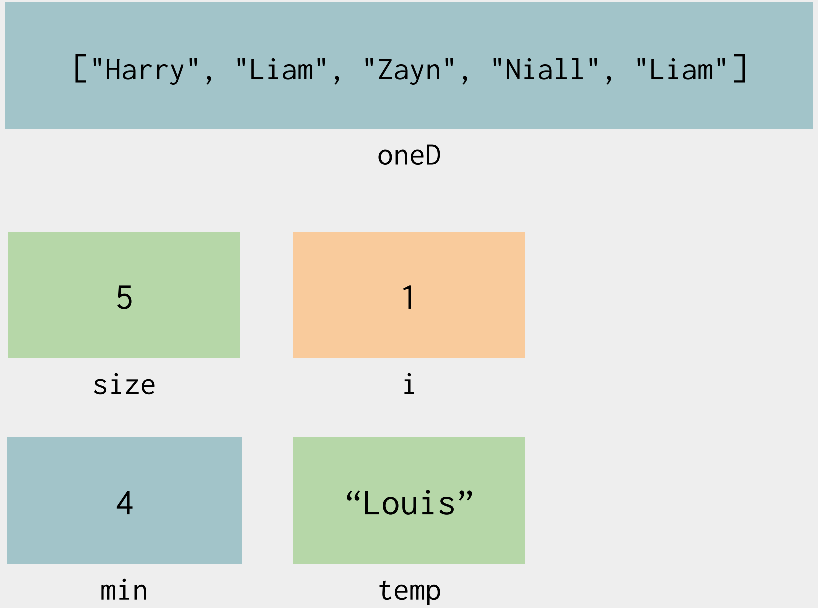
Walkthrough

```
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];
```



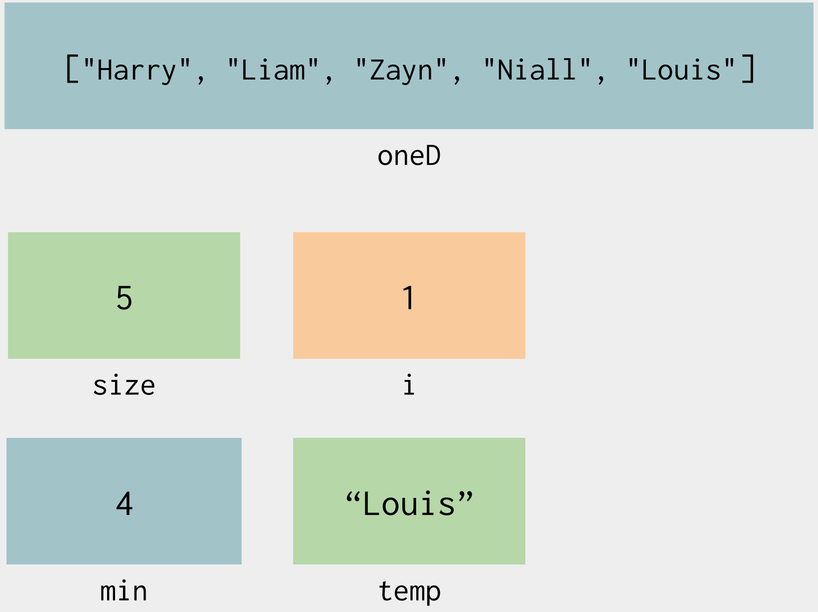
Walkthrough

```
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: 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 null 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>

void removeNonAlpha(char str[]) {
    for(int i = 0; str[i] != '\0'; i++)
        while ( !isalpha(str[i]) && str[i] != '\0' )
            for(int j = i; str[j] != '\0'; j++)
                str[j] = str[j+1];
}
```

Practice Question: Array Traversal w/ Pointers

Write a function that sums the items of an array of n integers using only pointers to traverse the array.

Solution: Array Traversal w/ Pointers

Simple array traversal, but with pointers. To get to item `i`, add `i` to your head pointer then dereference. This works because the compiler knows the size of an item in your array in C++. Sum values as you go by adding them to a `total` value created outside of the for loop.

```
int sum(int *head, int n) {
  int total = 0;
  for (int i = 0; i < n; i++) {
    total += *(head + i);
  }
  return total;
}
```

```
sum(arr, 5);
```

Practice Question: C String Reversal with Pointers

Implement the function `reverse`, which takes a C String as an argument and prints out the characters in reverse order. You are not allowed to use the `strlen` function, and you must use pointers in any traversal of the C String.

```
void reverse(const char s[]);

int main() {
     char str[] = “stressed”
     reverse(str);
     // OUTPUT: desserts
}
```

Solution: C String Reversal with Pointers

```
void reverse(const char s[]) {
     const char *p = s; // create a new pointer for our traversal
     while (*p != '\0') { // move the pointer to the end of the C String
          p++;
     }
     p--; // set p to point at the last char in the C String
     while (p >= s) { // print out chars as we traverse back to the beginning
          cout << *p;
          p--;
     }
     cout << endl;
}
```

Practice Question: strcat

Implement the C string concatenation function. The function takes two C strings and copies the chars from the source C string to the end of the destination C string. The original null byte of the destination is overwritten when copying the source. Return the destination pointer at the end of the function. You do not know the size of the destination and source C strings (so you can't create a temporary C string to store all of the characters!)

```
char* strcat(char* destination, const char* source);
```

Solution: strcat

```
char* strcat(char* destination, const char* source) {
    char* d = destination;
    while (*d) // this loop sets d to point at the null byte of destination
        d++;
    const char* s = source;
    while (*s) { // this loop copies the source C string to where d is pointing
        *d = *s;
        d++;
        s++;
    }
    *d = '\0'
    return destination; }
```

Good luck!

Sign-in	https://goo.gl/qSWgWN
MT 2	https://goo.gl/zJHeHV
MT 1	https://goo.gl/2FPFd8
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.