

UPE Tutoring:
CS 31 Midterm 1 Review

Sign-in <https://forms.gle/ita7uaMBtCTjcnmx9>

Slides link available upon sign-in



Table of Contents

- [Libraries & Namespaces](#)
- [Types & Variables](#)
- [Basic Input/Output](#)
- [C++ Strings](#)
- [If/Else](#)
- [Switches](#)
- [Loops](#)
 - [Pile of Money](#)
 - [String to Int](#)
 - [isPalindrome](#)
 - [Get “Switchy”](#)
- [Scoping](#)
- [Functions](#)
 - [Pass by Ref Practice](#)
- [Arrays](#)
 - [First Repeat Index](#)
 - [One Direction Sort](#)
 - [Transpose](#)
 - [Resolve Merge Issues](#)
- [Good Luck](#)



Libraries

- `#include` allows us to use a library
- `#include <iostream>` allows us to use things like:
 - `cin`
 - `cout`
 - `endl`
- Note: *iostream* stands for *input/output stream*



Namespaces

- using namespace std;
- A namespace is a collection of classes and functions
- If we don't call using namespace *ns_name*, we will have to specify the namespace of the function we want to call.
- e.g. std::cout, std::string, std::isdigit



Namespaces (cont.)

```
#include <iostream>

int main() {
    int age;
    std::cin >> age;
    std::cout << age;
    std::cout << std::endl;
}
```

```
#include <iostream>
using namespace std;

int main() {
    int age;
    cin >> age;
    cout << age;
    cout << endl;
}
```



Basic data types

- int, double, char
 - Declare variables to store values in memory
 - int x; // Creates a variable x of type int
 - char y; // Creates a variable y of type char
- Can initialize with value at declaration:
 - int a = 5;
 - double z = 53.24324;



Modifying variables

- The type of the variable must be specified only once, at the time of declaration

```
int x = 5;  
x = x + 5;  
x -= 6; // equivalent to x = x - 6;
```

```
double z;  
z = 53.234;  
z *= 5; // equivalent to z = z * 5;
```



Modifying variables (cont.)

- Integer division truncates after the decimal point
- The % (modulus) operator returns the remainder of integer division

```
int x = 5;  
int integerQuotient = x / 3; // integerQuotient equals 1  
int remainder = x % 3;      // remainder equals 2  
x %= 4;                   // same as x = x % 4, x now equals 1
```



Modifying variables (cont.)

- Double division
 - If at least one of the operands is a double, floating point division occurs.
 - If both values are integers, integer division occurs instead.

```
int x = 5;  
double unexpectedQuotient = x / 2;      // equals 2  
double expectedQuotient = x / 2.0;      // equals 2.5
```



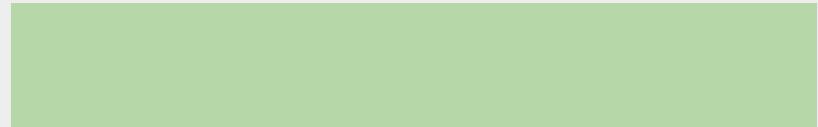
Input/output

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "How old are you? " << endl;
    cin >> age;
    cout << "You are " << age <<
        " years old" << endl;
}
```



Input/output

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "How old are you? " << endl;
    cin >> age;
    cout << "You are " << age <<
        " years old" << endl;
}
```



age



Input/output

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "How old are you? " << endl;
    cin >> age;
    cout << "You are " << age <<
        " years old" << endl;
}
```

> How old are you?



age



Input/output

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "How old are you? " << endl;
    cin >> age;
    cout << "You are " << age <<
        " years old" << endl;
}
```

> How old are you? 20

20

age



Input/output

```
#include <iostream>
using namespace std;
int main() {
    int age;
    cout << "How old are you? " << endl;
    cin >> age;
    cout << "You are " << age <<
        " years old" << endl;
}
```

> How old are you? 20
> You are 20 years old

20

age



Strings

- Used to store blocks of text
- Strings can be initialized from literals
 - `string x = "hello";`
- Individual characters can be accessed with the [] operator.
 - `char c = x[0]; // c == 'h'`



String operations

```
string x = "hello there";
```

- The `size()` method returns the number of characters in a string.
 - `int length = x.size(); // length equals 11`
- The `substr(startIndex, length)` method returns a substring *including* `startIndex` of length `length`.
 - `string sub = x.substr(3, 2); // sub equals "lo"`
- *Note: substr is not in the scope of the midterm.*



String operations

```
// The + operator is overloaded:  
// It appends to the end of strings.
```

```
int main() {  
    string x = "hello there";  
    x += ", my name is Mark";  
    cout << x << endl;  
}
```



hello there

x



String operations

```
// The + operator is overloaded:  
// It appends to the end of strings.
```

```
int main() {  
    string x = "hello there";  
    x += ", my name is Mark";  
    cout << x << endl;  
}
```



hello there, my name is Mark

x



String operations

```
// The + operator is overloaded:  
// It appends to the end of strings.  
  
int main() {  
    string x = "hello there";  
    x += ", my name is Mark";  
    cout << x << endl;  
}
```

```
> hello there, my name is Mark
```

hello there, my name is Mark

x



String input

```
// getline(...) consumes characters from  
// the input until it encounters a '\n'.  
  
int main() {  
    string x;  
    getline(cin, x);  
    cout << x << endl;  
}
```



String input

```
// getline(...) consumes characters from  
// the input until it encounters a '\n'.
```

```
int main() {  
    string x;  
    getline(cin, x);  
    cout << x << endl;  
}
```

```
> Why hello there!
```



String input

```
// getline(...) consumes characters from  
// the input until it encounters a '\n'.  
  
int main() {  
    string x;  
    getline(cin, x);  
    cout << x << endl;  
}
```

```
> Why hello there!  
> Why hello there!
```



Ignoring characters

- Undesirable characters are often left in the input buffer after using `cin`.
- `cin.ignore(int numChars, char delim)` can be used to “flush” out these undesired characters. It flushes up to the nearest `delim` or `numChar` characters, whichever comes first.
- `cin.ignore(...)` becomes necessary if after reading a number, the next thing you want to read is a string using `getline(...)`.



Ignoring characters

- Undesirable characters are often left in the input buffer after using `cin`.
- `cin.ignore(int numChars, char delim)` can be used to “flush” out these undesired characters. It flushes up to the nearest `delim` or `numChar` characters, whichever comes first.
- `cin.ignore(...)` becomes necessary if after reading a number, the next thing you want to read is a string using `getline(...)`.
- **Common question:** does `getline` consume '`\n`'?



Ignoring characters

- Undesirable characters are often left in the input buffer after using `cin`.
- `cin.ignore(int numChars, char delim)` can be used to “flush” out these undesired characters. It flushes up to the nearest `delim` or `numChar` characters, whichever comes first.
- `cin.ignore(...)` becomes necessary if after reading a number, the next thing you want to read is a string using `getline(...)`.
- **Common question:** does `getline` consume '`\n`'?
If a newline is found, it is extracted and discarded (i.e. it is not stored and the next input operation will begin after it).



`cin.ignore` example

```
int main() {
    cout << "How many Big Macs would you " <<
        "like? ";
    int bigMacs;
    cin >> bigMacs;
    cin.ignore(10000, '\n'); // Important!

    cout << "What else would you like " <<
        "with your order?";
    string sides;
    getline(cin, sides);
}
```

> How many Big Macs would you like?



Input

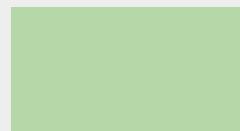


`cin.ignore` example

```
int main() {
    cout << "How many Big Macs would you " <<
        "like? ";
    int bigMacs;
    cin >> bigMacs;
    cin.ignore(10000, '\n'); // Important!

    cout << "What else would you like " <<
        "with your order?";
    string sides;
    getline(cin, sides);
}
```

> How many Big Macs would you like?



bigMacs



Input



cin.ignore example

```
int main() {
    cout << "How many Big Macs would you " <<
        "like? ";
    int bigMacs;
    cin >> bigMacs;
    cin.ignore(10000, '\n'); // Important!

    cout << "What else would you like " <<
        "with your order?";
    string sides;
    getline(cin, sides);
}
```

> How many Big Macs would you like? 1000

1000

bigMacs

\n

Input



cin.ignore example

```
int main() {
    cout << "How many Big Macs would you " <<
        "like? ";
    int bigMacs;
    cin >> bigMacs;
    cin.ignore(10000, '\n'); // Important!

    cout << "What else would you like " <<
        "with your order?";
    string sides;
    getline(cin, sides);
}
```

> How many Big Macs would you like? **1000**

1000

bigMacs



Input



UPSILON PI EPSILON

CS 31 Midterm 1 Review (Fall '19) shorturl.at/tNSW9

cin.ignore example

```
int main() {
    cout << "How many Big Macs would you " <<
        "like? ";
    int bigMacs;
    cin >> bigMacs;
    cin.ignore(10000, '\n'); // Important!

    cout << "What else would you like " <<
        "with your order?";
    string sides;
    getline(cin, sides);
}
```

- > How many Big Macs would you like? 1000
- > What else would you like with your order?

1000

bigMacs



Input



UPSILON PI EPSILON

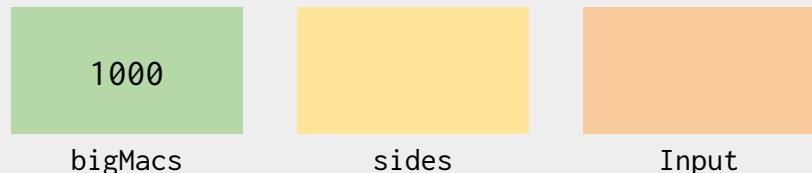
CS 31 Midterm 1 Review (Fall '19) shorturl.at/tNSW9

cin.ignore example

```
int main() {
    cout << "How many Big Macs would you " <<
        "like? ";
    int bigMacs;
    cin >> bigMacs;
    cin.ignore(10000, '\n'); // Important!

    cout << "What else would you like " <<
        "with your order?";
    string sides;
    getline(cin, sides);
}
```

- > How many Big Macs would you like? 1000
- > What else would you like with your order?

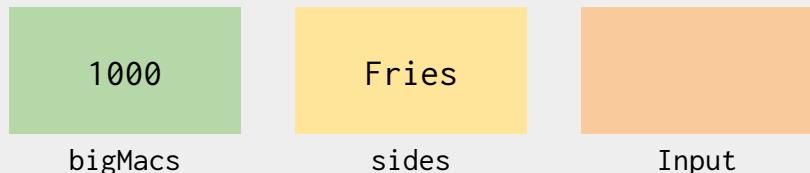


cin.ignore example

```
int main() {
    cout << "How many Big Macs would you " <<
        "like? ";
    int bigMacs;
    cin >> bigMacs;
    cin.ignore(10000, '\n'); // Important!

    cout << "What else would you like " <<
        "with your order?";
    string sides;
    getline(cin, sides);
}
```

> How many Big Macs would you like? 1000
> What else would you like with your
order? Fries



`cin.ignore(...)` example

- What will be stored in the string “a” in this example?
- Assume that input is newline terminated.

```
int x; string a;  
cout << "Enter an integer" << endl;  
cin >> x; // Assume the user enters "7"  
cout << "Enter a string" << endl;  
getline(cin, a); // Assume user enters "500"
```



cctype

- Useful shortcut methods for characters
- #include<cctype> gives you...
 - `isalpha('M')` // true, since 'M' is a letter
 - `isupper('M')` // true, since 'M' is an uppercase letter
 - `islower('r')` // true, since 'r' is a lowercase letter
 - `isdigit('5')` // true, since '5' is a digit character
 - `islower('M')` // false, since 'M' is not a lowercase letter
 - `isalpha(' ')` // false, since ' ' is not a letter
 - `isalpha('5')` // false, since '5' is not a letter



If statements

- if statements only run code if the condition is true
- *Note: any non-zero expression is considered true*

```
int age;  
cin >> age;  
if (age < 13) {  
    cout << "You are not yet a teenager!" << endl;  
}
```



If statements

- Without curly braces, only next statement is attached to the control statement.
- So, if you want multiple statements to be executed, use curly braces.
- *Note: this also applies to else and else-if statements*

```
if (cond1) {  
    statement1;  
    statement2;  
}
```



If statements (cont.)

```
int main() {  
    int x = 3;  
    if (x == 5)  
        cout << "x is 5" << endl;  
    cout << "In if" << endl; // Incorrectly  
                           // called!  
}
```

```
int main() {  
    int x = 5;  
    if (x == 5) {  
        cout << "x is 5" << endl;  
        cout << "In if" << endl;  
    }  
}
```



Else statements

- Performed when all if and else if conditions fail

```
int number;  
cin >> number;  
if (number % 2 == 0)  
    cout << "You gave an even number" << endl;  
else  
    cout << "You gave an odd number" << endl;
```



Else-if statements

- Allows us to check for more than the if condition and its complement

```
if (cond1)
    statement1;
else if (cond2)
    statement2;
else if (cond3)
    statement3;
else
    statement4;
```



Comparison pitfalls

- **Equals-equals (==) vs. Equals (=)**
- These operators are very different!

`(x == y) // Returns true if x and y are equal`

`(x = y) // Assigns the value of y to x and returns the value
// ASSIGNED to x.`



Conditional confusion?

- Does this output anything?

```
int age = 17;  
if (age) {  
    cout << "You are not 0 years old!" << endl;  
}
```



Conditional confusion?

- What does this output?

```
int age = 0;  
if (age) {  
    cout << "You are not 0 years old!" << endl;  
} else {  
    cout << "You are 0 years old!" << endl;  
}
```



Switches

- Arguably a more compact alternative to long if/else if/else sequences
- The value tested must be an integral type or convertible to one
 - e.g. int, char, short, long, etc.
 - string is not a permitted type
- A break statement must be used to leave the switch. Otherwise execution will fall through to the next case.



Switches (cont.)

```
string value; int number;  
cin >> number;  
switch (number) {  
    case 0: // Fall-through to Case 2.  
    case 2:  
        value = "Good";  
        break; // Remember to break!  
    case 3:  
        value = "Bad";  
        break;  
    default:  
        value = "Ugly";  
        break;  
}
```

Common question 1: is a break statement required for the default case?



Switches (cont.)

```
string value; int number;  
cin >> number;  
switch (number) {  
    case 0: // Fall-through to Case 2.  
    case 2:  
        value = "Good";  
        break; // Remember to break!  
    case 3:  
        value = "Bad";  
        break;  
    default:  
        value = "Ugly";  
        break;  
}
```

Common question 1: is a break statement required for the default case?

No, if the default case is at the end. However, we recommend that you put one anyways. This allows the default case to appear in a different order, and not necessarily at the end of the switch statement.



Switches (cont.)

```
string value; int number;  
cin >> number;  
switch (number) {  
    case 0: // Fall-through to Case 2.  
    case 2:  
        value = "Good";  
        break; // Remember to break!  
    case 3:  
        value = "Bad";  
        break;  
    default:  
        value = "Ugly";  
        break;  
}
```

Common question 1: is a break statement required for the default case?

No if the default case is at the end. However, we recommend that you put one anyways. This allows the default case to appear in a different order, and not necessarily at the end of the switch statement.

Common question 2: do I need a default statement?



Switches (cont.)

```
string value; int number;  
cin >> number;  
switch (number) {  
    case 0: // Fall-through to Case 2.  
    case 2:  
        value = "Good";  
        break; // Remember to break!  
    case 3:  
        value = "Bad";  
        break;  
    default:  
        value = "Ugly";  
        break;  
}
```

Common question 1: is a break statement required for the default case?

No if the default case is at the end. However, we recommend that you put one anyways. This allows the default case to appear in a different order, and not necessarily at the end of the switch statement.

Common question 2: do I need a default statement?

No, but it is good to have to catch unexpected cases. You should leave a //comment if you don't have a default to explain why!



While loops

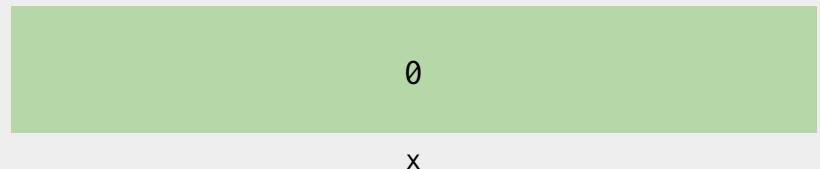
- while loops run code until the condition is false

```
int count;  
cin >> count;  
while (count >= 0) {  
    cout << "Countdown: " << count << endl;  
    count--;  
}
```



While loops

```
int main() {
    int x = 0;
    while (x < 2) {
        cout << x << endl;
        x++;
    }
    cout << "Done!" << endl;
}
```



While loops

```
int main() {
    int x = 0;
    while (x < 2) {
        cout << x << endl;
        x++;
    }
    cout << "Done!" << endl;
}
```

> 0



While loops

```
int main() {
    int x = 0;
    while (x < 2) {
        cout << x << endl;
        x++;
    }
    cout << "Done!" << endl;
}
```

> 0



While loops

```
int main() {
    int x = 0;
    while (x < 2) {
        cout << x << endl;
        x++;
    }
    cout << "Done!" << endl;
}
```

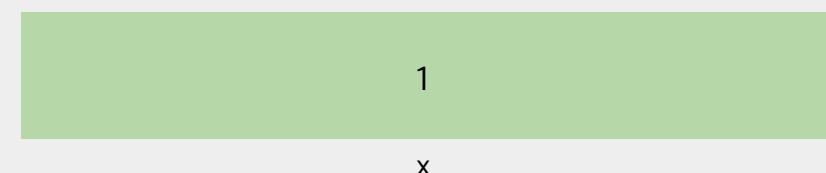
> 0



While loops

```
int main() {
    int x = 0;
    while (x < 2) {
        cout << x << endl;
        x++;
    }
    cout << "Done!" << endl;
}
```

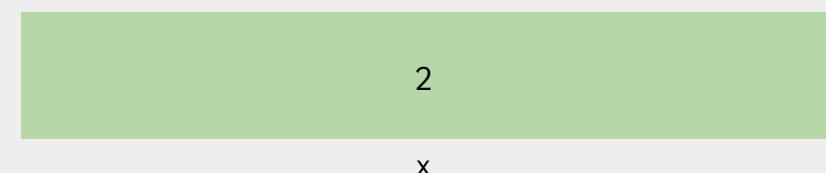
> 0
> 1



While loops

```
int main() {
    int x = 0;
    while (x < 2) {
        cout << x << endl;
        x++;
    }
    cout << "Done!" << endl;
}
```

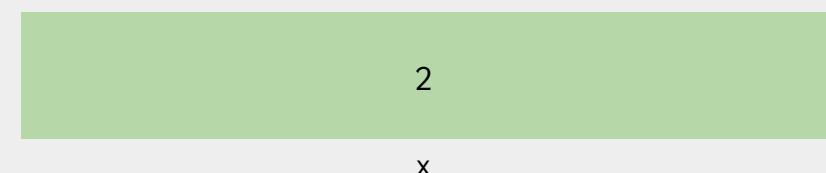
> 0
> 1



While loops

```
int main() {
    int x = 0;
    while (x < 2) {
        cout << x << endl;
        x++;
    }
    cout << "Done!" << endl;
}
```

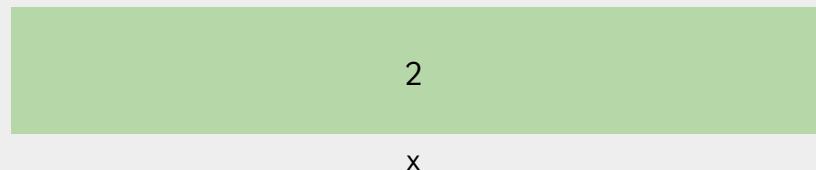
> 0
> 1



While loops

```
int main() {  
    int x = 0;  
    while (x < 2) {  
        cout << x << endl;  
        x++;  
    }  
    cout << "Done!" << endl;  
}
```

> 0
> 1
> Done!



Do-while loops

- Same as while loops, except the first iteration always runs.

```
statement1;  
do {  
    statement2;  
} while (cond1); // Don't forget the semicolon!  
  
statement3;
```



For loops

- Declaration is run once before anything else
- Condition is evaluated before the code block is executed
- Action is run after the code block is executed

```
for (declaration; condition; action) {  
    statement1;  
    statement2;  
}
```

Note: all of the three sections of the for loop are optional; all that is required is the semicolon. If condition is empty, it defaults to always true. Example: `for(int i = 0;;i++) { //infinite loop }`



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

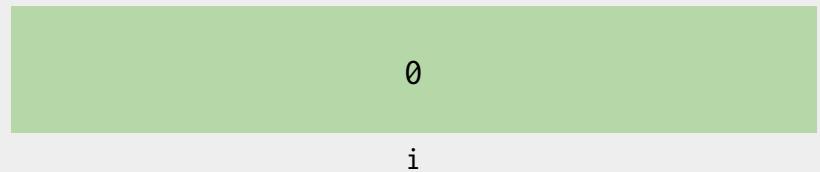
    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```

> i is now equal to: 0

0

i



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```

> i is now equal to: 0

1
i



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```

> i is now equal to: 0

1

i



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
> i is now equal to: 1
```

1

i



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
> i is now equal to: 1
```

2

i



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
> i is now equal to: 1
```

2

i



For loops

```
int main() {
    for (int i = 0; i < 2; i++) {
        cout << "i is now equal to: " << i <<
            endl;
    }

    // Note that i is now out of scope.
    cout << "Done!" << endl;
}
```

```
> i is now equal to: 0
> i is now equal to: 1
> Done!
```



Nested loops

```
for (int i = 1; i <= 10; i++) {  
    for (int j = 1; j <= 10; j++) {  
        cout << (i * j) << "\t";  
    }  
    cout << endl;  
}
```



Quick Question - Breaking the Outer Loop

- What happens when you break inside nested loops?



Quick Question - Breaking the Outer Loop

- What happens when you break inside nested loops?
 - Only the loop that contains the break statement is broken out of.



Quick Question - Breaking the Outer Loop

- What happens when you break inside nested loops?
 - Only the loop that contains the break statement is broken out of.
- Solution: use a boolean variable (a *flag*) in your loop's statements and change the boolean from true to false when you want to break out of a nested loop.

```
bool keepLoopingI = true;  
for (int i = 1; i <= 100; i++) {  
    for (int j = 1; j <= 200; j++) {  
        if (i+j > 250) keepLoopingI = false;           What does this print?  
        cout << i << "," << j << endl;  
    }  
    if (!keepLoopingI) break;  
}
```



Practice Question: Pile of Money

Print a pile of money n stacks high, leaning up on the right against your mansion's wall.

Example for $n=4$:

```
$|  
$$|  
$$$|  
$$$$|
```

(Contributed by Matthew Wong)



Solution: Pile of Money

```
int n = 5; // Or any positive integer value

// Loop through all n rows
for (int i = 1; i <= n; i++) {
    // print the spaces preceding the dollar signs
    for (int spaces = 0; spaces < n - i; spaces++)
        cout << " ";

    // print the i dollar signs for the row
    for (int money = 0; money < i; money++)
        cout << "$";

    // print the wall and move to the next line
    cout << "|" << endl;
}
```



Practice Question: String to Int

Convert a variable of type `string` into the integer represented by that string.

Example, convert:

```
    string number = "125"      into  
    int numberAsInt // holding the value 125 as an integer
```

Hints:

```
// string name = "Daniel";  
// name[0] is 'D'  
// name.size() is 6  
// '8' - '0' (the character eight minus the character zero) is 8 (the integer eight)
```

(Contributed by Katie Luangkote)



Solution: String to Int

```
string number = "125"; // or let the user cin their own string
int result = 0;
int multiplier = 1;
for (int i = number.size() - 1; i >= 0; i--) {
    result += (number[i] - '0') * multiplier;
    multiplier *= 10;
}
int numberAsInt = result;

// "125" becomes 5 + 2*10 + 1*100 = 125
// Any other way to do this?
```



Practice Question: isPalindrome

Write a function `is_palindrome` that takes a string as an argument and returns true if the string is a **palindrome** and false if it is not. A palindrome is a string that is read backwards the same way as it is forwards. For example "racecar" backwards is "racecar".



Solution: isPalindrome

Start by checking that the first and the last letters in the string are equal, then move inward until we reach the middle of the string. Note: This only requires $n / 2$ iterations of the for-loop. Also, this code works even when n is **odd**. Trace through an example by hand to see why!

```
bool is_palindrome(string s) {  
    int n = s.size();  
    for (int i = 0; i < n / 2; i++) {  
        int j = (n - 1) - i;  
        if (s[i] != s[j]) return false;  
    }  
    return true;  
}
```



Practice Question: Get “Switchy”

```
int main() {  
    string morty;  
    int rick = 5;  
    rick = (rick + 2 * 5) / 10;  
    switch (rick) {  
        case 1:  
            morty = "lubba";  
            break;  
        case 3:  
            morty = "aw geez";  
            break;  
        default:  
            morty = "oh man";  
            break;  
    }  
    ...  
    ...  
    morty += morty;  
    morty[0] = 'w';  
    for (int i = 0; i < 2; i++)  
        morty += "dub";  
    cout << morty << endl;  
}
```

What is the output of this code?



Solution: Get “Switchy”

```
int main() {  
    string morty;  
    int rick = 5;  
    rick = (rick + 2 * 5) / 10; rick = 1  
    switch (rick) {  
        case 1:  
            morty = "lubba"; morty = "lubba"  
            break;  
        case 3:  
            morty = "aw geez";  
            break;  
        default:  
            morty = "oh man";  
            break;  
    }  
    ...  
    ...  
    morty += morty; morty = "lubbalubba"  
    morty[0] = 'w'; morty = "wubbalubba"  
  
    for (int i = 0; i < 2; i++)  
        morty += "dub"; morty = "wubbalubbadubdub"  
  
    cout << morty << endl;  
}
```

Output: wubbalubbadubdub

Translation: I am in great pain please help me



Scoping

- Variables only exist within the curly brackets or the implied curly brackets that they were written in.

```
if (cond1) {  
    statement1;  
}
```



Scoping (cont.)

```
if (cond1) {  
    int x = 5;  
    cout << x << endl; // No error  
}  
  
cout << x << endl; // Error!! x doesn't exist  
                      // outside the if statement
```



Scoping (cont.)

```
int x = 1;  
if (cond1) {  
    x = 5;  
    cout << x << endl; // No error  
}  
  
cout << x << endl; // No error here either!
```



Scoping (cont.)

```
string s1 = "bonjour";
for (int i = 0; i < s1.size(); i++) {
    char lastChar = s1[i];
}

// Both i and lastChar don't exist here!
cout << i << " " << lastChar << endl; // Error!
```



Scoping (cont.)

```
string s1 = "bonjour";
int i; char lastChar;
for (i = 0; i < s1.size(); i++) {
    lastChar = s1[i];
}

// Now both i and lastChar exist here
cout << i << " " << lastChar << endl;
```



Scoping: Switch Statements

```
int main() {
    int n;
    cin >> n;
    switch (n) {
        case 1:
            int x = 10;
            cout << "You entered 1! 1 times 10 is " << x << endl;
            break;
        default:
            int x = 5;
            cout << "You didn't enter 1" << endl;
    }
}
```



Scoping: Switch Statements

Warning: the cases of a switch statement share the same scope!

```
int main() {  
    int n;  
    cin >> n;  
    switch (n) {  
        case 1:  
            int x = 10;  
            cout << "You entered 1! 1 times 10 is " << x << endl;  
            break;  
        default:  
            int x = 5; // This is an error! Compiler says “error: redefinition of 'x'”  
            cout << "You didn't enter 1" << endl;  
    }  
}
```



Scoping: Switch Statements

Warning: the cases of a switch statement share the same scope!

```
int main() {
    int n;
    cin >> n;
    switch (n) {
        case 1:
            int x = 10;
            cout << "You entered 1! 1 times 10 is " << x << endl;
            break;
        default:
            // Does x exist here? It's within the switch's curly braces, but "int x = 10" was never executed!?
            cout << "You didn't enter 1" << endl;
    }
}
```



Scoping: Switch Statements

Warning: the cases of a switch statement share the same scope!

```
int main() {  
    int n;  
    cin >> n;  
    switch (n) {  
        case 1:  
            int x = 10;  
            cout << "You entered 1! 1 times 10 is " << x << endl;  
            break;  
        default:  
            // So, this is also an error! Compiler says "note: jump bypasses variable initialization"  
            cout << "You didn't enter 1" << endl;  
    }  
}
```



Scoping: Switch Statements

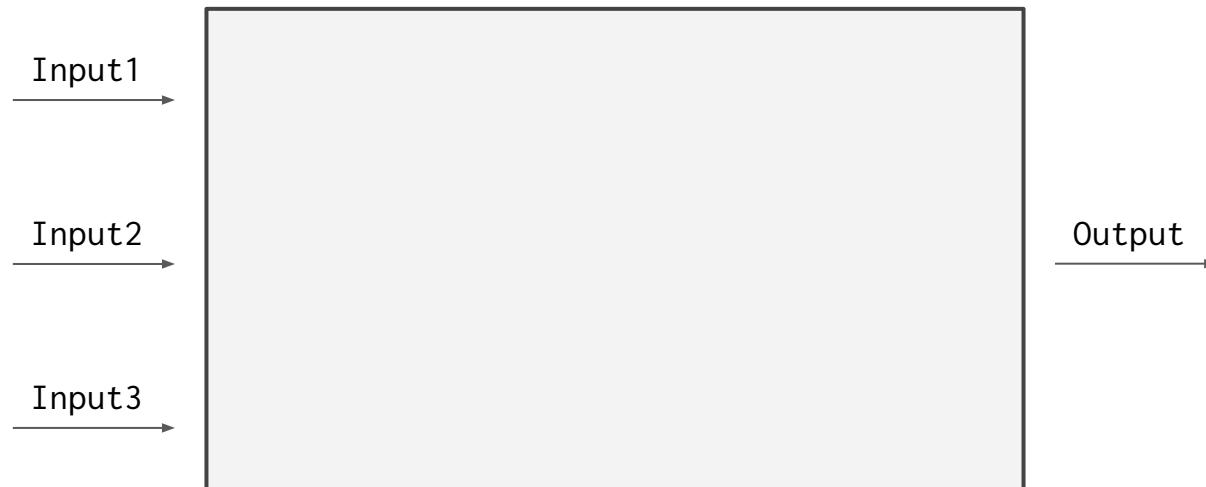
Warning: the cases of a switch statement share the same scope!

```
int main() {
    int n;
    cin >> n;
    switch (n) {
        case 1: {
            int x = 10; // Now x is only known to the scope of this case
            cout << "You entered 1! 1 times 10 is " << x << endl;
            break;
        }
        default:
            cout << "You didn't enter 1" << endl;
    }
}
```



Functions: The Box Model

functionName



Functions: Simple Example

```
#include <iostream>
using namespace std;

int hypotenuse(int side1, int side2) {
    /* Function body */
    /* We don't need to know how the function is implemented */
    ...
}

int main() {
    int x = hypotenuse(3,4);
}
```



Functions: Mathematical Example

- $f(x, y, z) = x - y - z;$
- $f(10, 5, 4) = 10 - 5 - 4 = 1$
- $f(3, 5, 4) = 3 - 5 - 4 = -6$
- Symbolically, x, y, and z are input1, input2, and input3



Math Example (cont.)

Suppose we define $f(x, y, z) = x - y - z$.

```
int main() {  
    int b = 5;  
    int a = 4;  
    int c = 6;  
    cout << f(c,b,a) << endl;  
}
```

> -3

Then for some variables a, b, c , we have $f(c, b, a) = c - b - a = 6 - 5 - 4 = -3$.



Mathematical Example (cont.)

```
#include <iostream>
using namespace std;
int functionName(int num1, int num2, int num3) {
    return num1 - num2 - num3;
}
```

- $f(x, y, z)$ becomes `functionName(num1, num2, num3)`
- $\text{functionName}(10, 5, 4) = 10 - 5 - 4 = 1;$
- $\text{functionName}(3, 5, 4) = 3 - 5 - 4 = -6;$



Functions: Scoping

- Variables declared outside the function do not exist inside the function unless they are global variables

```
const int foo = 6;
string functionName(int a, int b) {
    cout << x << endl; // x does not exist here, so this is an ERROR.
    int y = 5 + foo;   // This is okay because foo is global.
    return y + a + b;
}
```



Functions: Scoping (cont.)

Variables declared inside the function do not exist outside the function

```
int main() {  
    int x = 4;  
    cout << functionName(4,5) << endl;  
    cout << y << endl; // y does not exist here, so this is an error!  
}
```



Functions: Nested Loops Example

Problem: Find the first character in string1 that exists in string2, or return the null byte if no such character is found.

Examples:

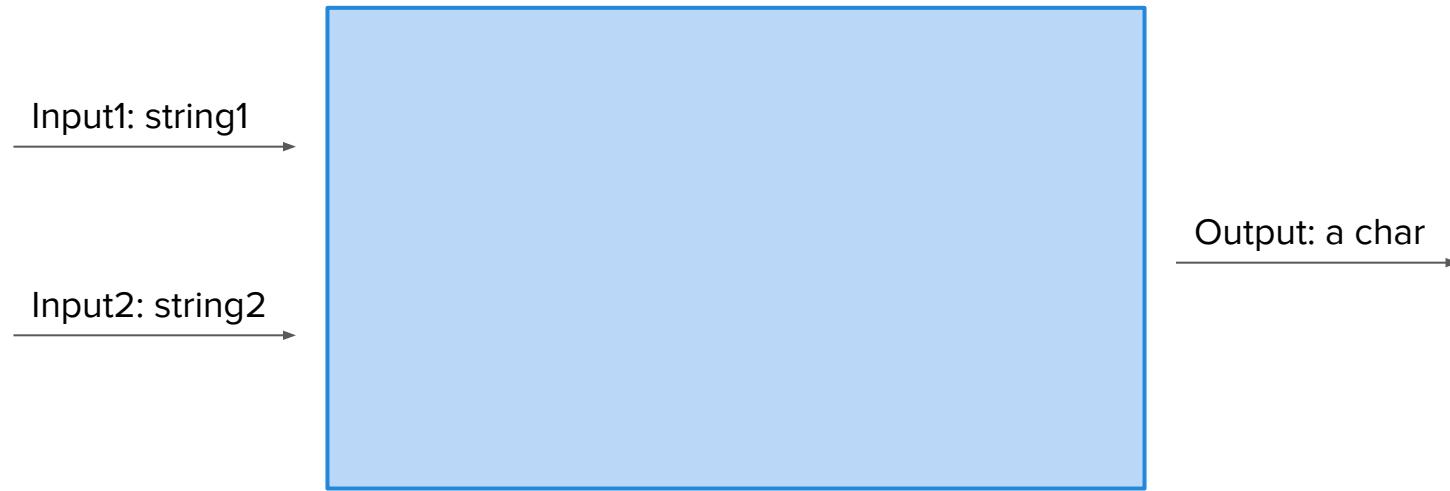
- string1 = “hello”, string2 = “there”, result = ‘h’
- string1 = “aabc”, string2 = “xyzab”, result = ‘a’
- string1 = “aabc”, string2 = “xyzzb”, result = ‘b’
- string1 = “abcd”, string2 = “wxyz”, result = ‘\0’



Functions: The Box Model

```
char firstChar(string string1, string string2);
```

firstChar



Functions: Nested Loops Solution

```
char firstChar(string string1, string string2) {  
    for (int i = 0; i < string1.size(); i++)  
        for (int j = 0; j < string2.size(); j++)  
            if (string1[i] == string2[j])  
                return string1[i];  
    return '\0';  
}
```



Functions: Calling the firstChar Function

```
int main() {  
    cout << firstChar("hello", "there") << endl;  
    cout << firstChar("aabc", "xyzab") << endl;  
    cout << firstChar("aabc", "xyzzb") << endl;  
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}
```

```
int main() {
    cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}

int main() {
    cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}

int main() {
    cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}

int main() {
    cout << firstChar("hello", "there");
}
```

"hello" "there"
string1 string2

'h'
string1[i]

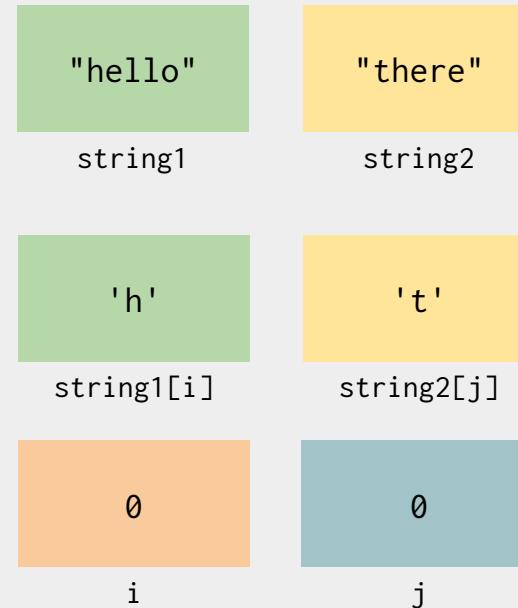
0
i



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}

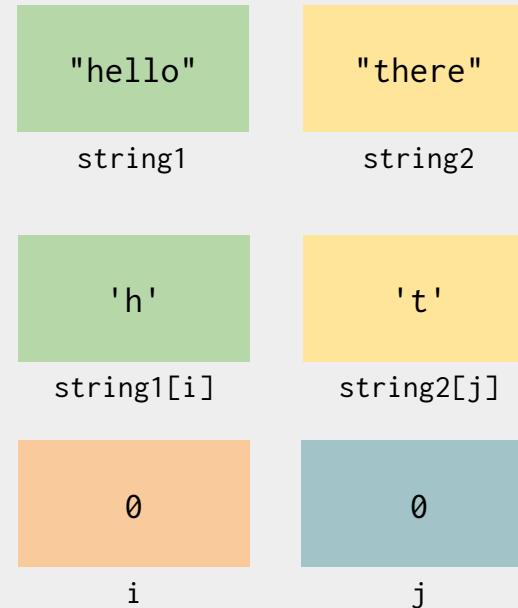
int main() {
    cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}

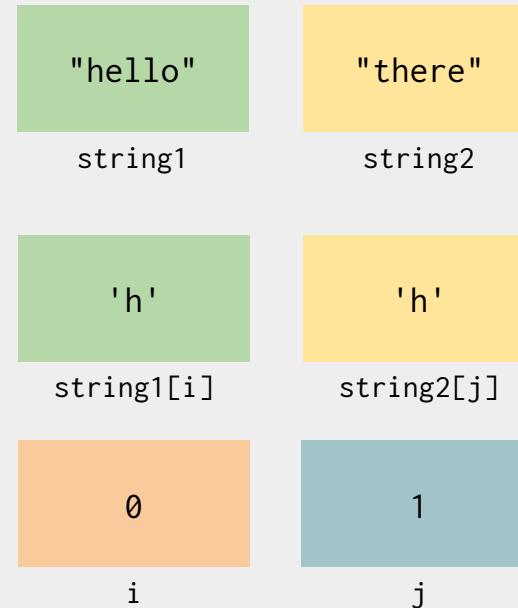
int main() {
    cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}

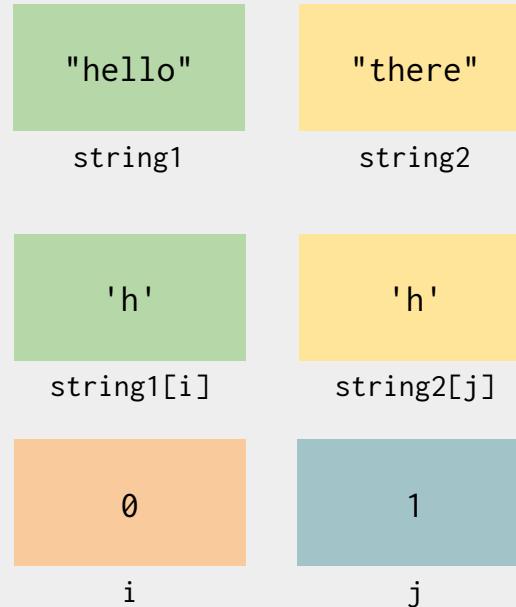
int main() {
    cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}

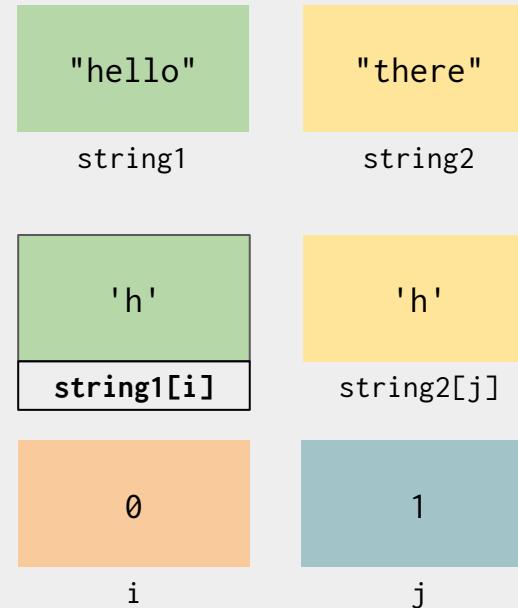
int main() {
    cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,
    string string2) {
    for (int i=0; i < string1.size(); i++)
        for (int j=0; j < string2.size(); j++)
            if (string1[i] == string2[j])
                return string1[i];
    return '\0';
}

int main() {
    cout << firstChar("hello", "there");
}
```



Walkthrough

```
char firstChar(string string1,  
    string string2) {  
    for (int i=0; i < string1.size(); i++)  
        for (int j=0; j < string2.size(); j++)  
            if (string1[i] == string2[j])  
                return string1[i];  
    return '\0';  
}  
  
int main() {  
    cout << firstChar("hello", "there");  
}
```

> h



Walkthrough

```
char firstChar(string string1,  
    string string2) {  
    for (int i=0; i < string1.size(); i++)  
        for (int j=0; j < string2.size(); j++)  
            if (string1[i] == string2[j])  
                return string1[i];  
    return '\0';  
}  
  
int main() {  
    cout << firstChar("hello", "there");  
}
```

> h



Functions: Parameters

- The types, modifiers, order, and number of parameters are all important in a function declaration
 - types: string, int, bool, etc.
 - modifiers: &, const, *, etc.
 - number: how many parameters are passed to a particular function?
- Function call **must** match the pattern of the declaration



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(string s1, string s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    string x = isEqual(s1,s2, position);  
}
```



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(string s1, string s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    string x = isEqual(s1,s2, position);  
}
```

No. The return type is boolean.



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(string s1, string s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    bool x = isEqual(s1,s2);  
}
```



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(string s1, string s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    bool x = isEqual(s1,s2);  
}
```

No. The number of arguments doesn't match.



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(string s1, string s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    bool x = isEqual(position, s1, s2);  
}
```



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(string s1, string s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    bool x = isEqual(position, s1, s2);  
}
```

No. The order of arguments doesn't match.



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(string s1, string s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    bool x = isEqual(s1, s2, position);  
}
```



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(string s1, string s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    bool x = isEqual(s1, s2, position);  
}
```

Yes.



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(const string& s1, const string& s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    bool x = isEqual(s1, s2, position);  
}
```



Will this compile?

```
// Assume this function is defined later.  
bool isEqual(const string& s1, const string& s2, int position);  
  
int main() {  
    string s1 = "hello";  
    string s2 = "there";  
    int position = 1;  
    bool x = isEqual(s1, s2, position);  
}
```

Yes. Notice the argument passing syntax is identical for pass by 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

```
void swap (int& x, int& y);
```



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 the 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.
- What if 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

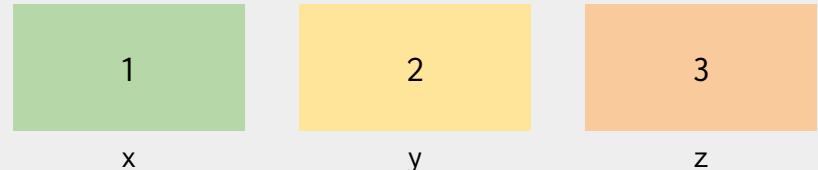
What does this program print out?

```
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

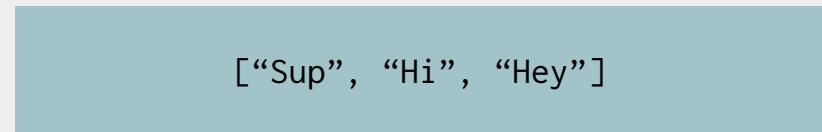
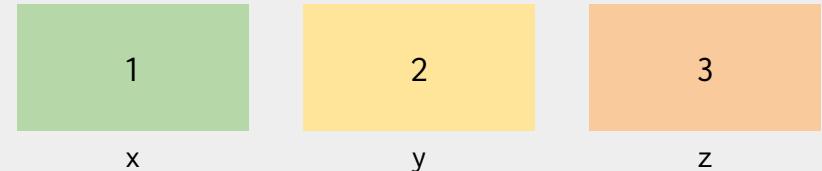
```
int main() {  
    int x = 1, y = 2, z = 3;  
    string arr[] = {"Sup", "Hi", "Hey"};  
  
    // Print values before the call to foo  
    cout << x << " " << y << " " << z << endl;  
    // Print the return value of foo  
    cout << foo(x, y, z, arr) << endl;  
    // Print values after the call to foo  
    cout << x << " " << y << " " << z <<  
        endl;  
}
```



Walkthrough

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

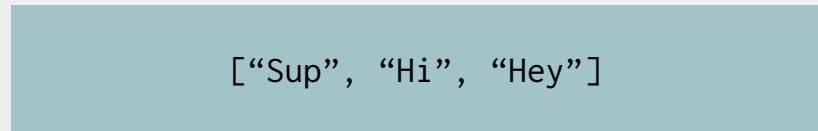
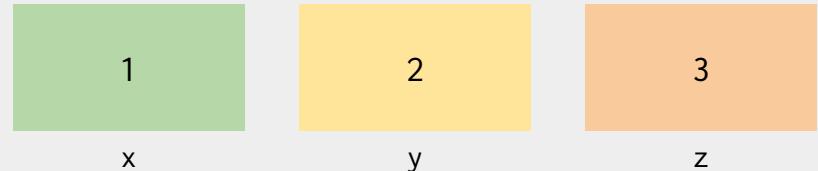
    // Print values before the call to foo
    cout << x << " " << y << " " << z << endl;
    // Print the return value of foo
    cout << foo(x, y, z, arr) << endl;
    // Print values after the call to foo
    cout << x << " " << y << " " << z <<
        endl;
}
```



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;
}
```

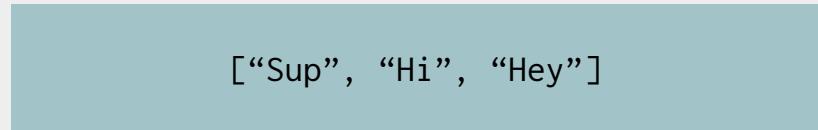
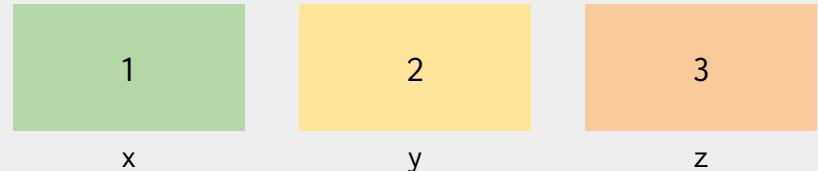


> 1 2 3



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;  
}
```

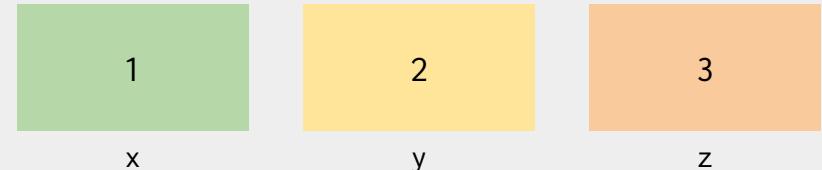


> 1 2 3



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;  
}
```

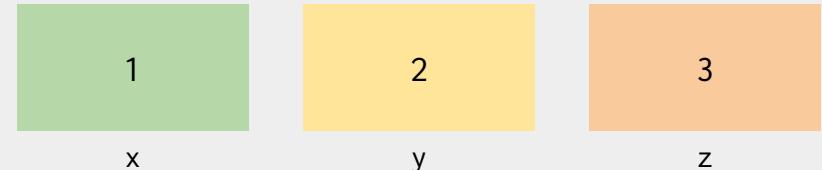


> 1 2 3



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;  
}
```

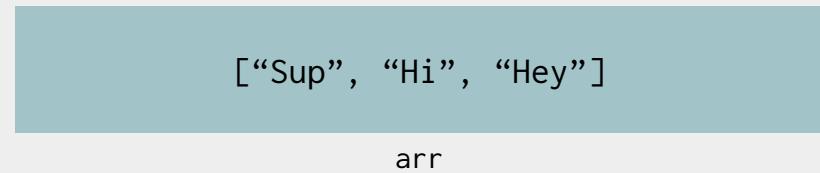
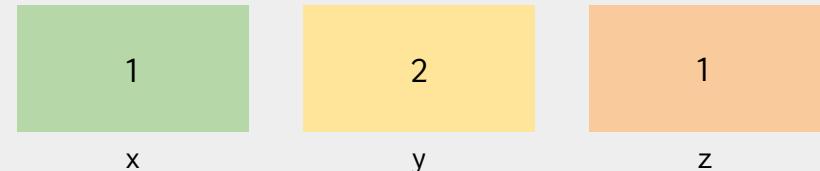


```
> 1 2 3
```



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;  
}
```

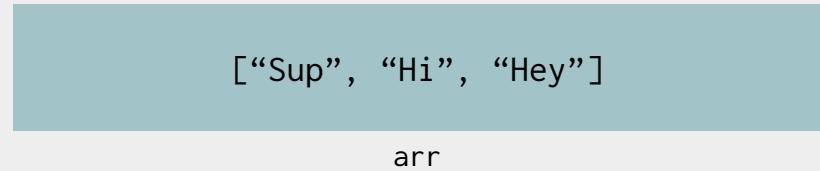
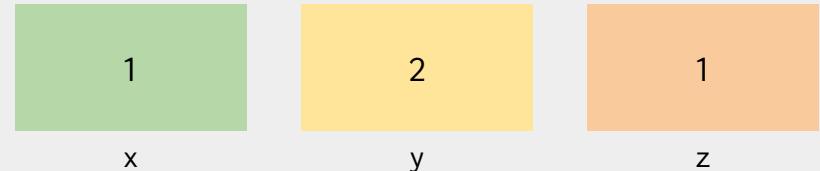


> 1 2 3



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;  
}
```

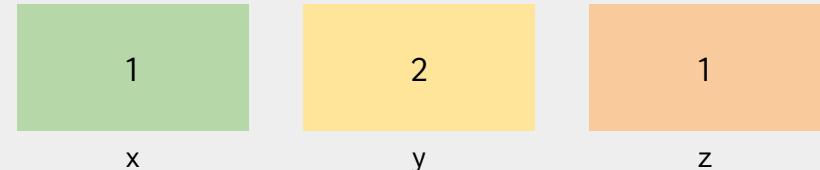


```
> 1 2 3  
> 1 2 1
```



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;  
}
```

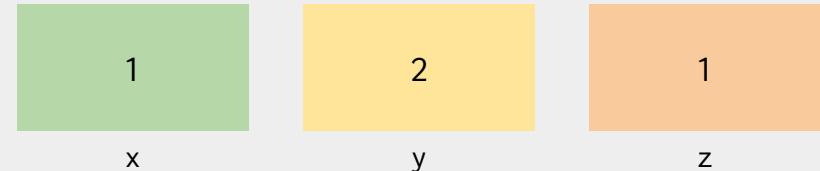


```
> 1 2 3  
> 1 2 1
```



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;  
}
```

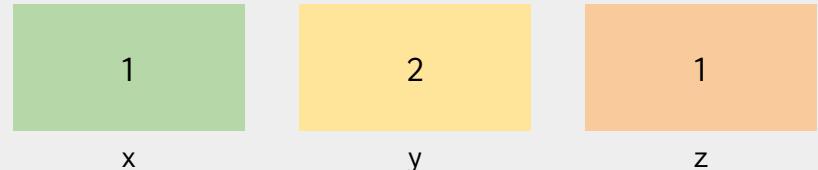


```
> 1 2 3  
> 1 2 1
```



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;  
}
```

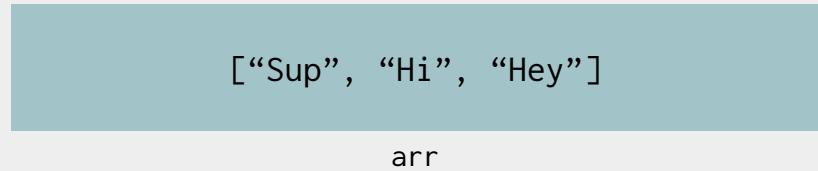
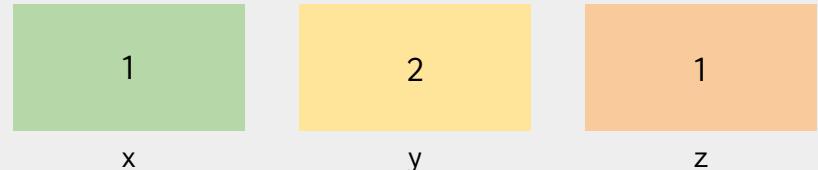


```
> 1 2 3  
> 1 2 1
```



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;  
}
```

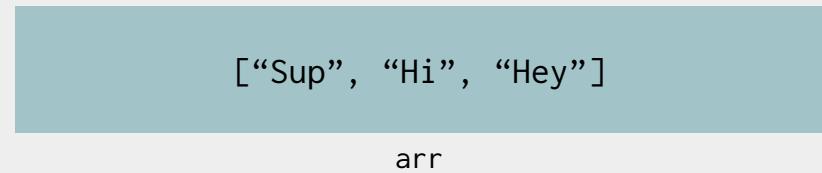
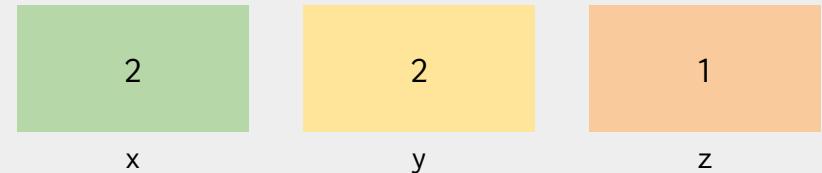


```
> 1 2 3  
> 1 2 1
```



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;  
}
```

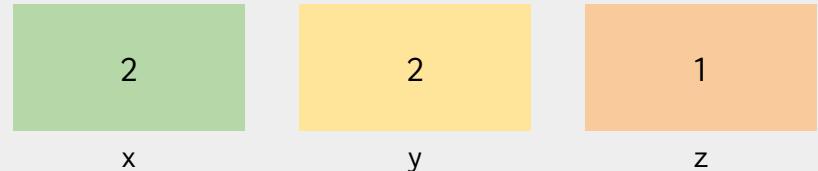


```
> 1 2 3  
> 1 2 1
```



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;  
}
```

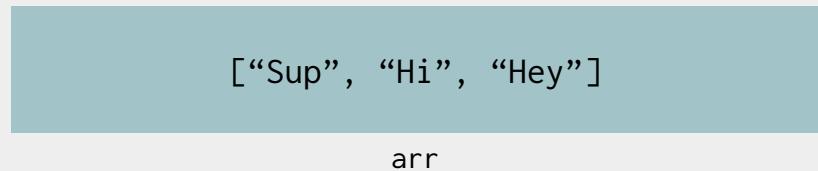
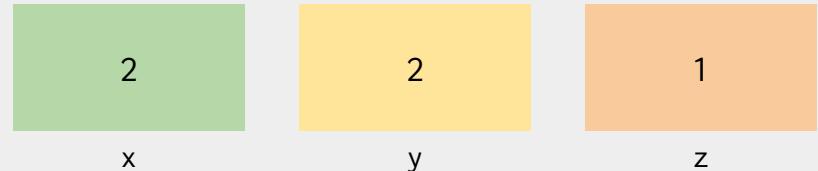


```
> 1 2 3  
> 1 2 1  
> 2 2 1
```



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;  
}
```

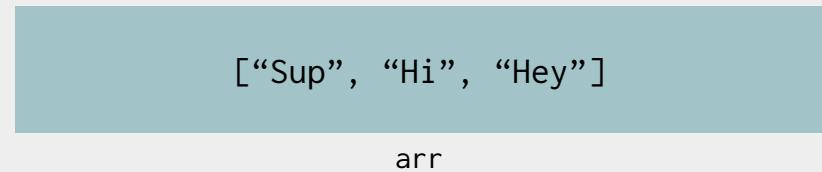
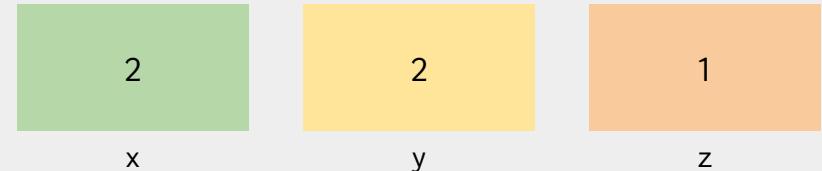


```
> 1 2 3  
> 1 2 1  
> 2 2 1
```



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;  
}
```

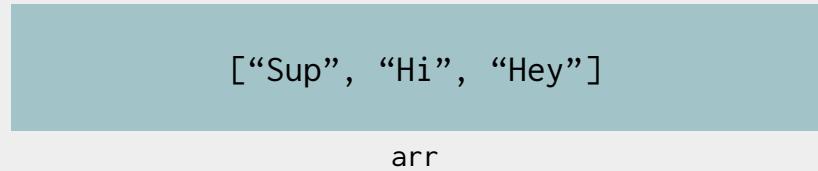
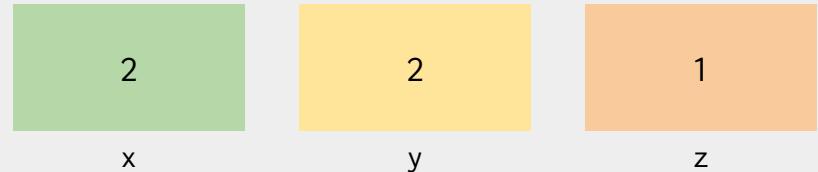


```
> 1 2 3  
> 1 2 1  
> 2 2 1
```



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;  
}
```

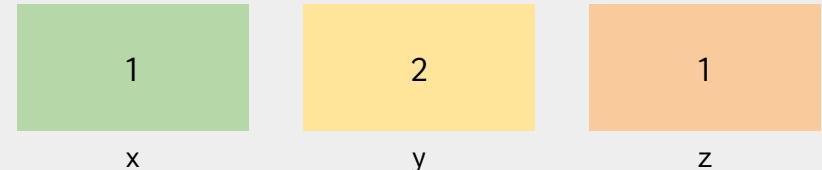


```
> 1 2 3  
> 1 2 1  
> 2 2 1
```



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;  
}
```

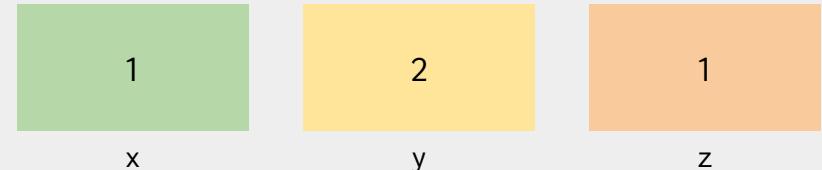


```
> 1 2 3  
> 1 2 1  
> 2 2 1
```



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;  
}
```

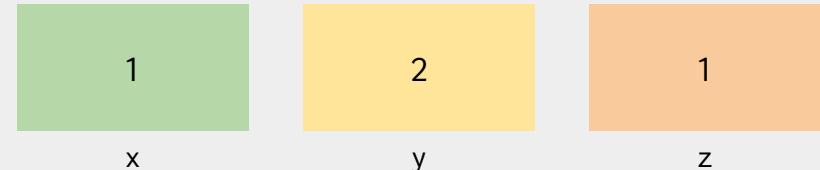


```
> 1 2 3  
> 1 2 1  
> 2 2 1
```



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;  
}
```

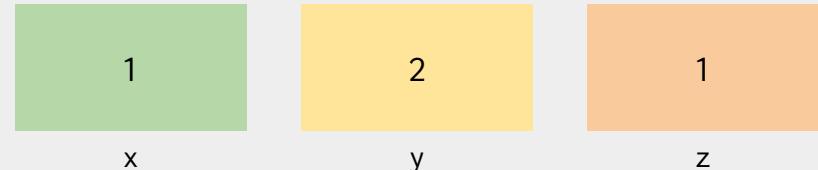


```
> 1 2 3  
> 1 2 1  
> 2 2 1
```



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;  
}
```

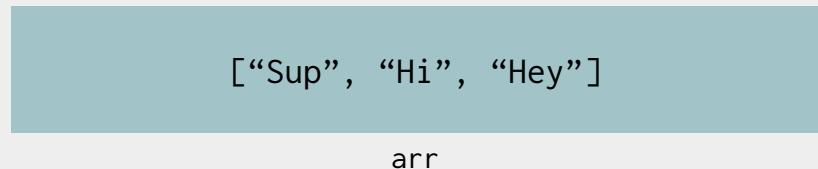
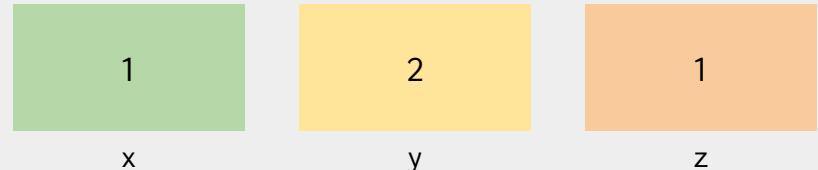


```
> 1 2 3          > 0  
> 1 2 1          > 1  
> 2 2 1
```



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



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;  
}
```

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;  
}
```

5

n

[2, 6, 3, 5, 10]

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;  
}
```

5

n

[2, 6, 3, 5, 10]

arr

5

n

0

count



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

0

i



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

[2, 6, 3, 5, 10]
arr

5 0 0
n count i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

1

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

1

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

1

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

2

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

2

i



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

[2, 6, 3, 5, 10]
arr

5 0 2
n count i



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

[2, 6, 2, 5, 10]

arr

5

n

0

count

2

i



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

[2, 6, 2, 5, 10]

arr

5

n

1

count

2

i



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

[2, 6, 2, 5, 10]
arr

5 1 3
n count i



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

[2, 6, 2, 5, 10]

arr

5

n

1

count

3

i



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

[2, 6, 2, 5, 10]

arr

5

n

1

count

3

i



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

[2, 6, 2, 4, 10]

arr

5

n

1

count

3

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

3

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

4

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

4

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

4

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

5

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

5

i



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

[2, 6, 2, 4, 10]

arr

6

n

2

count

5

i



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

[2, 6, 2, 4, 10]

arr

6

n

2

count

5

i



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

[2, 6, 2, 4, 10]

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;  
}
```

5

n

[2, 6, 2, 4, 10]

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 **n**th 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 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?**

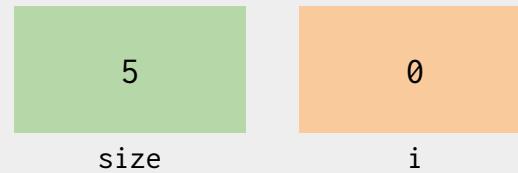


Walkthrough

```
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];
```

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

oneD

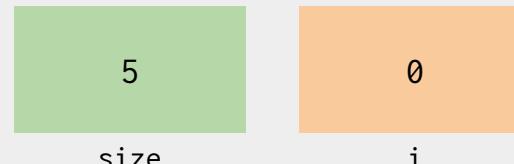


Walkthrough

```
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];
```

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

oneD



5

size

0

i

0

min



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];
```

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

oneD



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];
```

["Zayn", "Louis", "Harry", "Niall", "Liam"]

oneD



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];
```

["Zayn", "Louis", "Harry", "Niall", "Liam"]

oneD

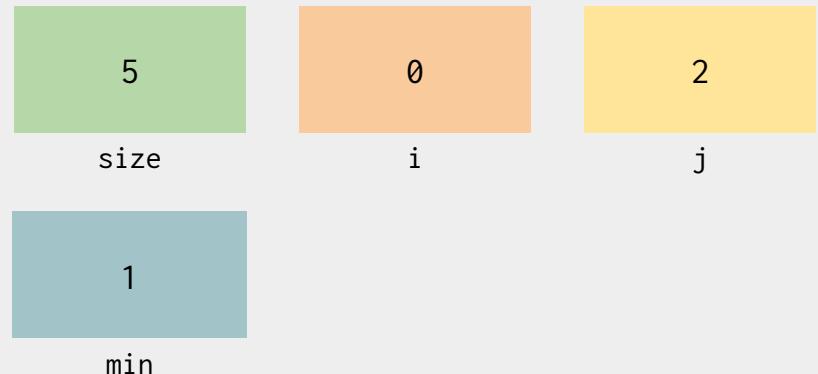


Walkthrough

```
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];
```

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

oneD

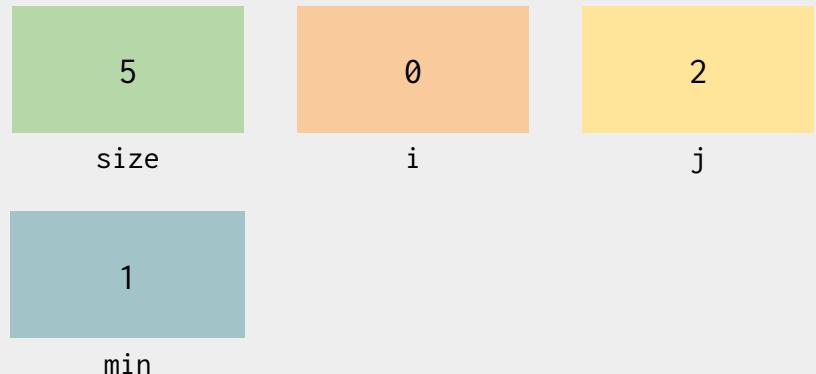


Walkthrough

```
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];
```

["Zayn", "Louis", "Harry", "Niall", "Liam"]

oneD



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];
```

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

oneD



5

size

0

i

2

j

2

min

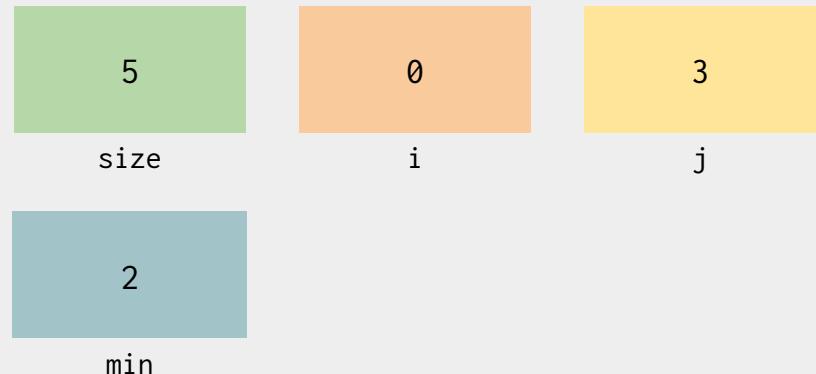


Walkthrough

```
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];
```

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

oneD

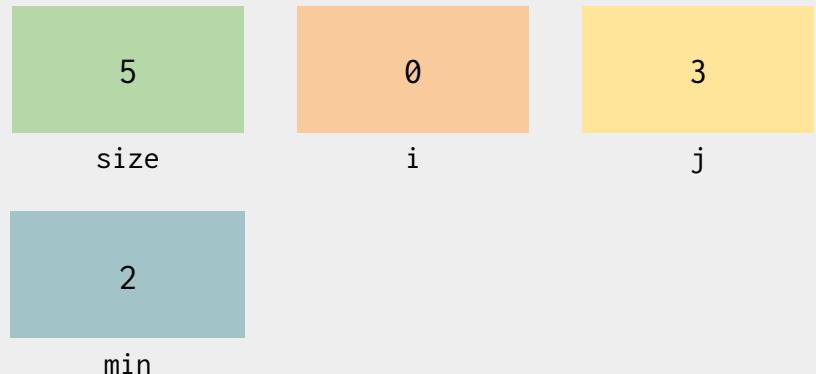


Walkthrough

```
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];
```

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

oneD

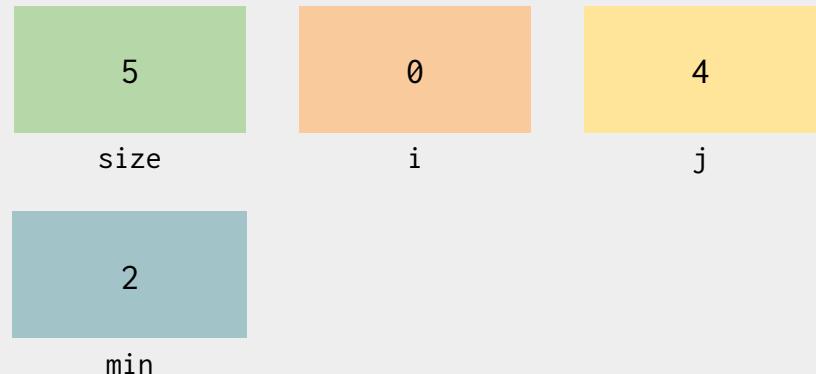


Walkthrough

```
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];
```

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

oneD

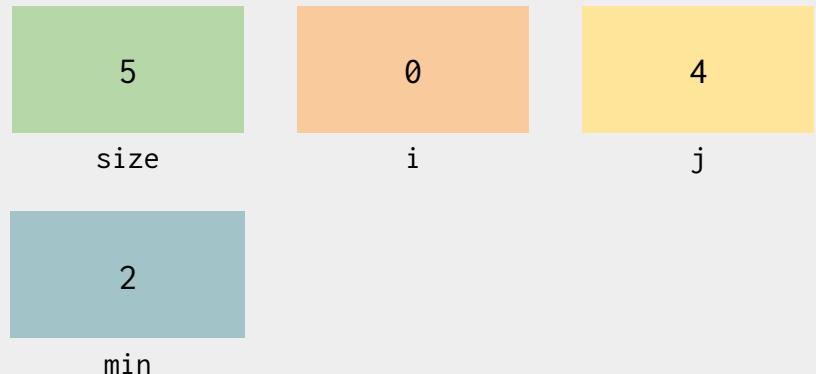


Walkthrough

```
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];
```

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

oneD

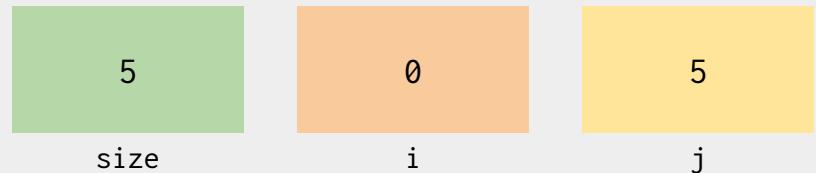


Walkthrough

```
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];
```

["Zayn", "Louis", "Harry", "Niall", "Liam"]

oneD



2

min

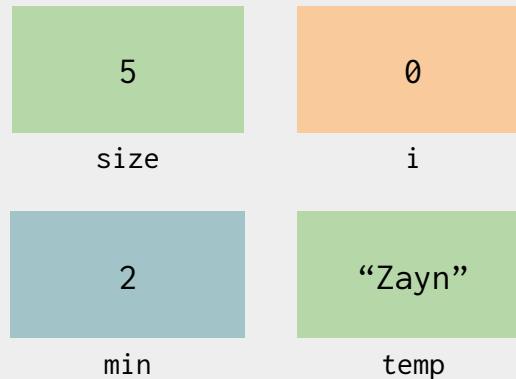


Walkthrough

```
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];
```

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

oneD



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];
```

```
["Harry", "Louis", "Harry", "Niall", "Liam"]
```

oneD



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];
```

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

oneD

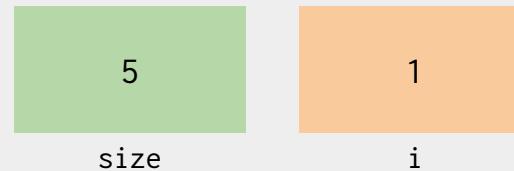


Walkthrough

```
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];
```

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

oneD

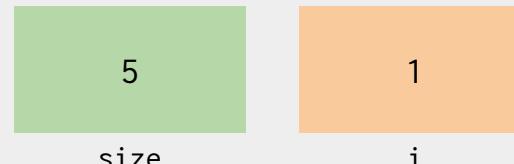


Walkthrough

```
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];
```

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

oneD

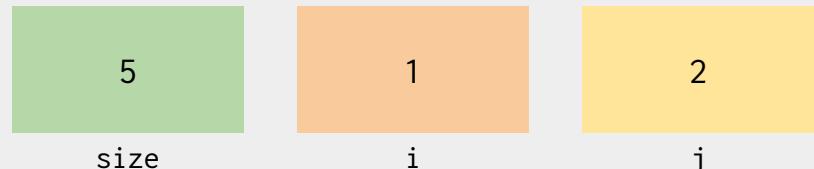


Walkthrough

```
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];
```

["Harry", "Louis", "Zayn", "Niall", "Liam"]

oneD

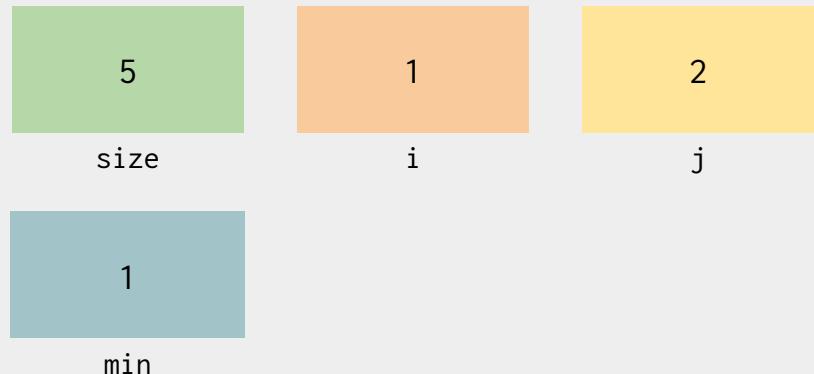


Walkthrough

```
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];
```

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

oneD

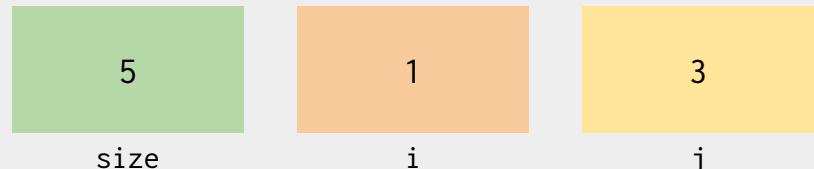


Walkthrough

```
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];
```

["Harry", "Louis", "Zayn", "Niall", "Liam"]

oneD



min

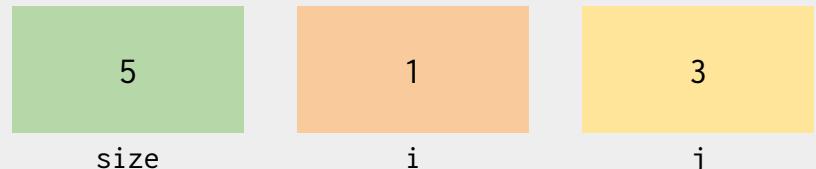


Walkthrough

```
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];
```

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

oneD

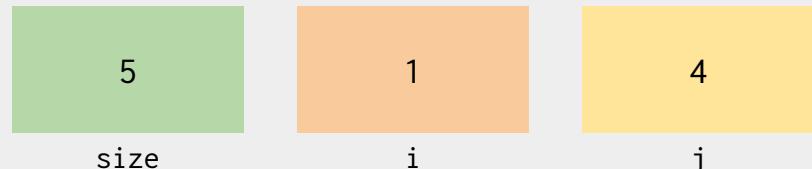


Walkthrough

```
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];
```

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

oneD



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];
```

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

oneD



1
min

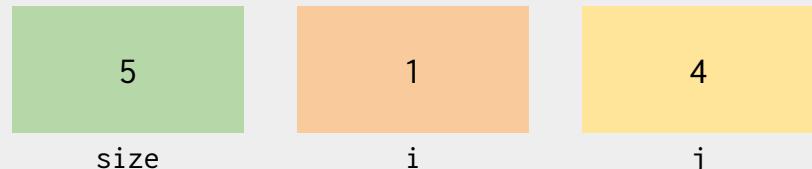


Walkthrough

```
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];
```

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

oneD



4

min

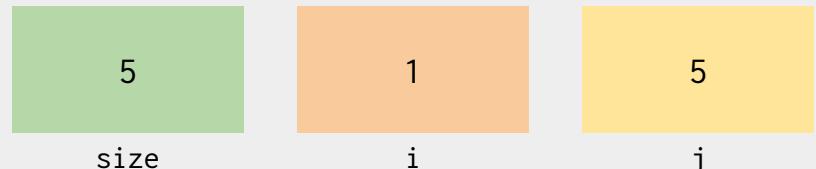


Walkthrough

```
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];
```

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

oneD



5

size

1

i

5

j

4

min

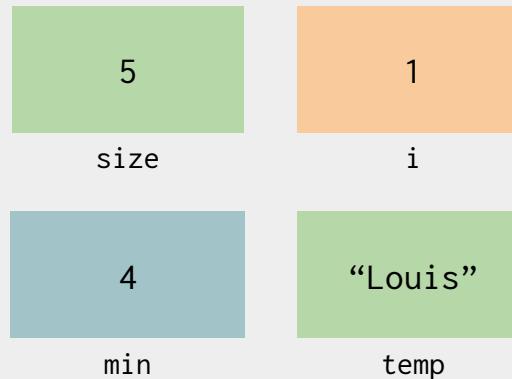


Walkthrough

```
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];
```

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

oneD

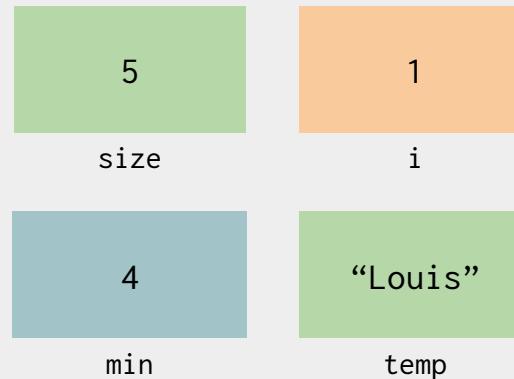


Walkthrough

```
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];
```

```
["Harry", "Liam", "Zayn", "Niall", "Liam"]
```

oneD

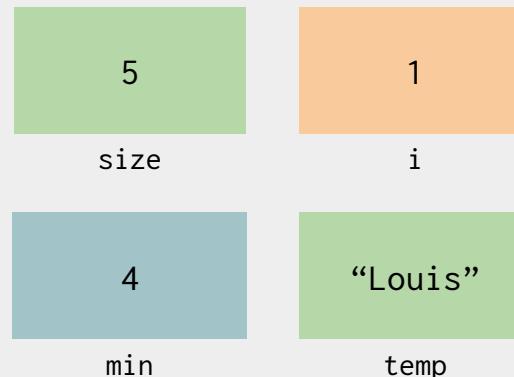


Walkthrough

```
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];
```

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

oneD



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: Transpose

Implement a function that takes in a pointer to an $n \times n$ 2d array of ints and transposes it. That is, the rows should become the columns and vice versa.

```
void transpose(int** matrix, int n);
```

Example: 1 2 3 ($n = 3$) \rightarrow 1 4 7
 4 5 6 2 5 8
 7 8 9 3 6 9



Solution: Transpose

```
void transpose(int** matrix, int n) {  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < i; j++) {  
            int temp = matrix[i][j];  
            matrix[i][j] = matrix[j][i];  
            matrix[j][i] = temp;  
        }  
    }  
}
```



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++;
}
}
```



Good luck!

Sign-in <https://forms.gle/ita7uaMBtCTjcnmx9>

Slides <https://shorturl.at/tNSW9>

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.



UPE Tutoring:
CS 31 Midterm 2 Review

Sign-in <http://bit.ly/2QCmwRH>

Slides link available upon sign-in



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?

5

n

```
// 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

[2, 6, 3, 5, 10]

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;  
}
```

5

n

[2, 6, 3, 5, 10]

arr

5

n

0

count



UPSILON PI EPSILON

CS 31 Midterm 2 Review (Fall '19)

<http://bit.ly/2QCmwRH>

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

[2, 6, 3, 5, 10]

arr

5

n

0

count

0

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

0

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

1

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

1

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

1

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

2

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

2

i



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

[2, 6, 3, 5, 10]

arr

5

n

0

count

2

i



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

[2, 6, 2, 5, 10]

arr

5

n

0

count

2

i



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

[2, 6, 2, 5, 10]

arr

5

n

1

count

2

i



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

[2, 6, 2, 5, 10]

arr

5

n

1

count

3

i



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

[2, 6, 2, 5, 10]

arr

5

n

1

count

3

i



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

[2, 6, 2, 5, 10]

arr

5

n

1

count

3

i



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

[2, 6, 2, 4, 10]

arr

5

n

1

count

3

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

3

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

4

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

4

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

4

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

5

i



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

[2, 6, 2, 4, 10]

arr

5

n

2

count

5

i



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

[2, 6, 2, 4, 10]

arr

6

n

2

count

5

i



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

[2, 6, 2, 4, 10]

arr

6

n

2

count

5

i



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

[2, 6, 2, 4, 10]

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;  
}
```

5

n

[2, 6, 2, 4, 10]

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 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?**

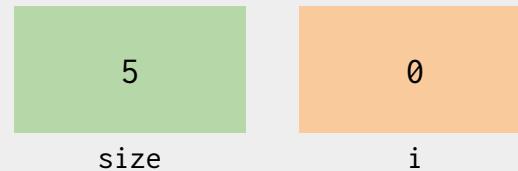


Walkthrough

```
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];
```

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

oneD

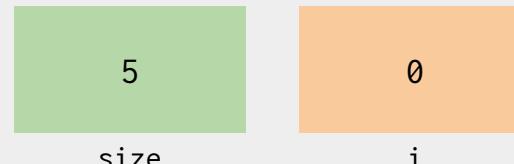


Walkthrough

```
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];
```

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

oneD



5

size

0

i

0

min



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];
```

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

oneD

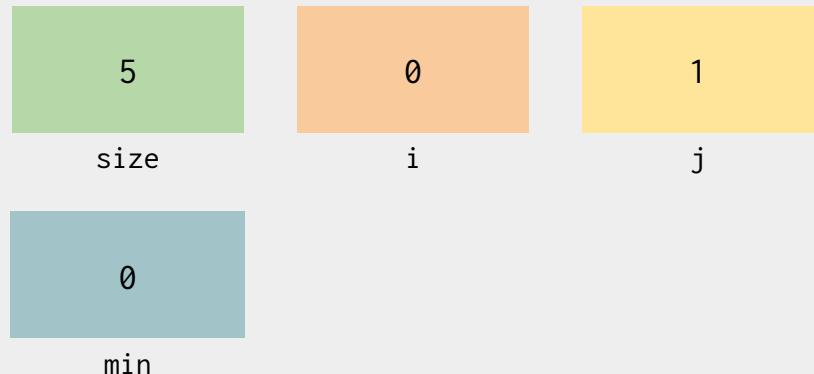


Walkthrough

```
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];
```

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

oneD

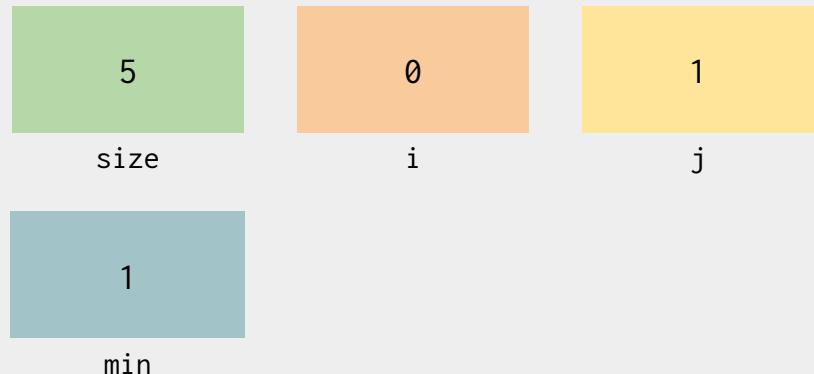


Walkthrough

```
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];
```

["Zayn", "Louis", "Harry", "Niall", "Liam"]

oneD

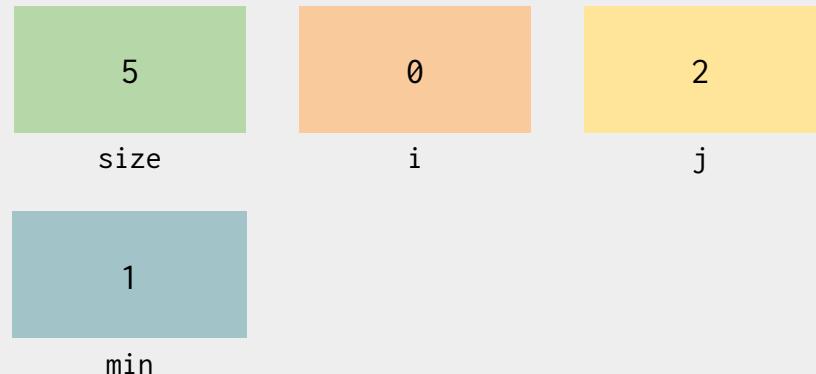


Walkthrough

```
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];
```

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

oneD

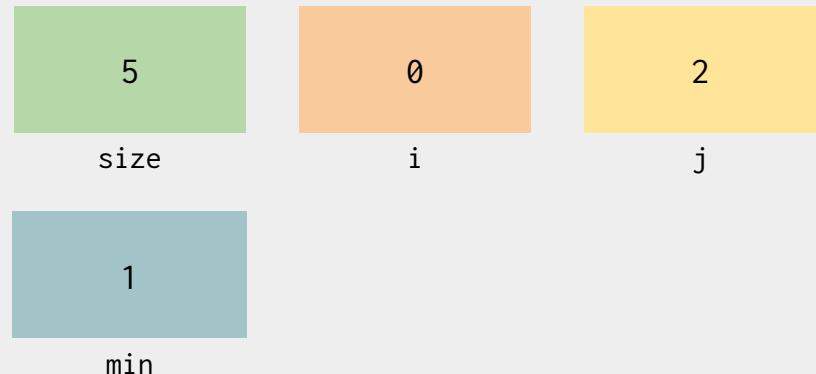


Walkthrough

```
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];
```

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

oneD



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];
```

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

oneD



5

size

0

i

2

j

2

min

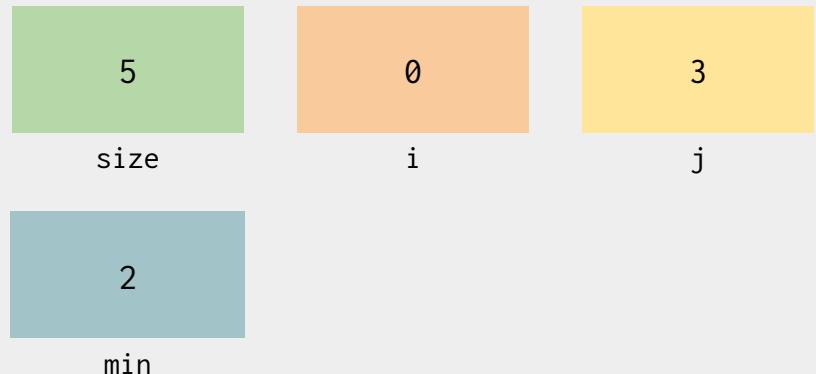


Walkthrough

```
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];
```

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

oneD

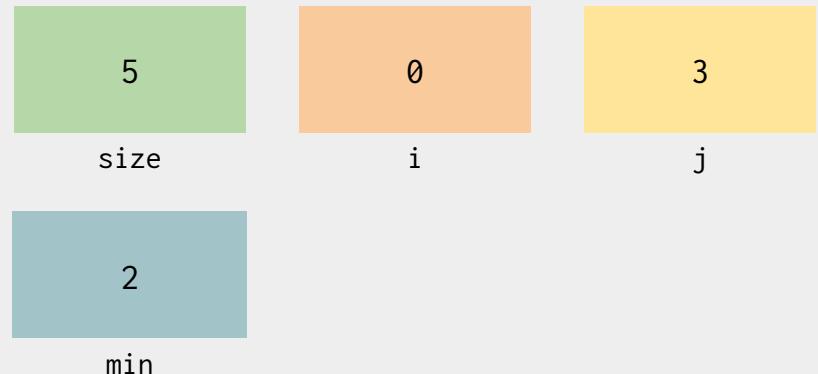


Walkthrough

```
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];
```

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

oneD

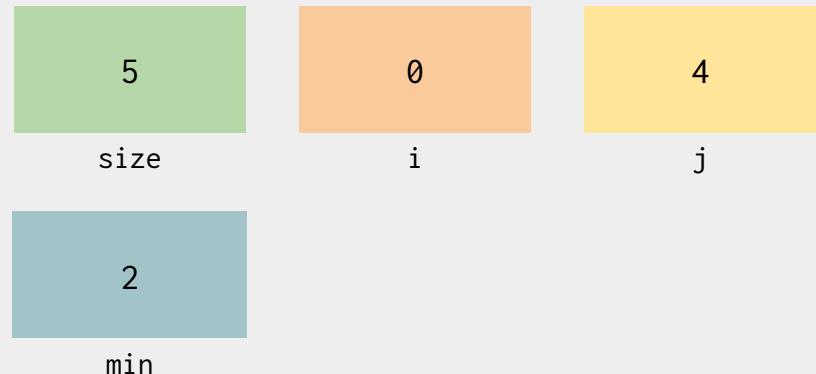


Walkthrough

```
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];
```

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

oneD

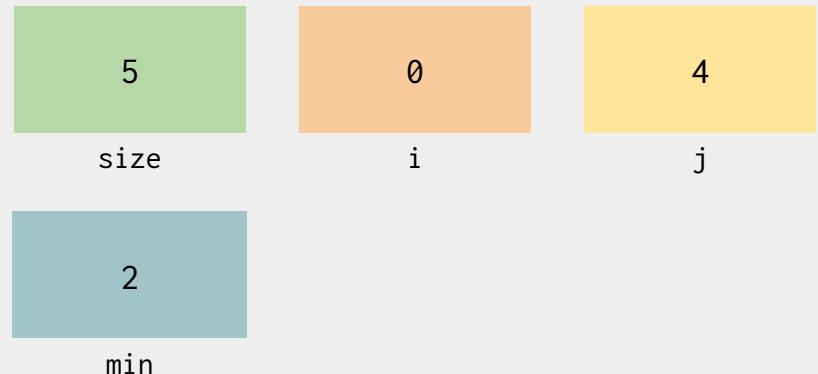


Walkthrough

```
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];
```

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

oneD



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];
```

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

oneD



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];
```

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

oneD

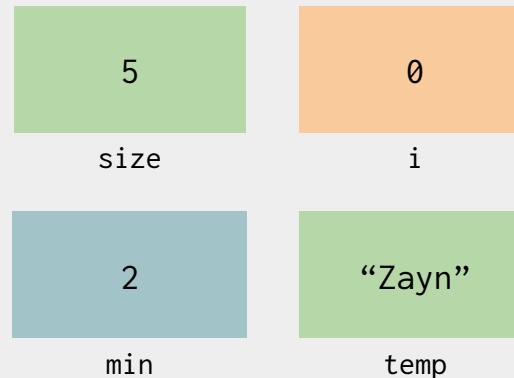


Walkthrough

```
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];
```

["Harry", "Louis", "Harry", "Niall", "Liam"]

oneD

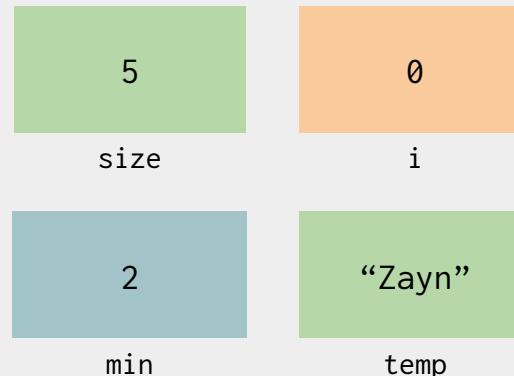


Walkthrough

```
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];
```

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

oneD

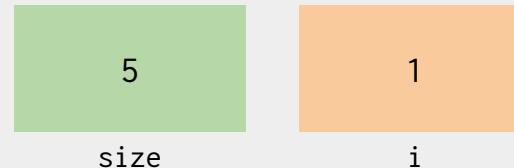


Walkthrough

```
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];
```

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

oneD

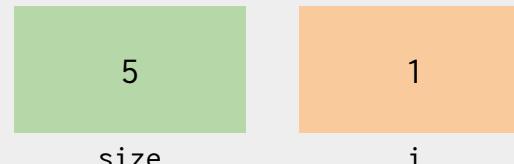


Walkthrough

```
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];
```

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

oneD



size
i

min

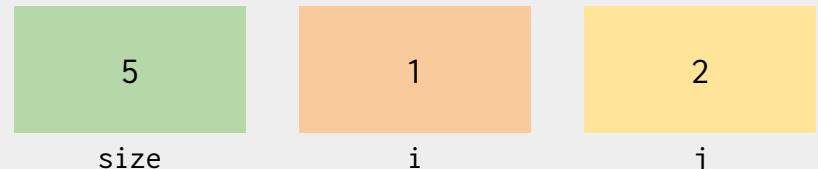


Walkthrough

```
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];
```

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

oneD

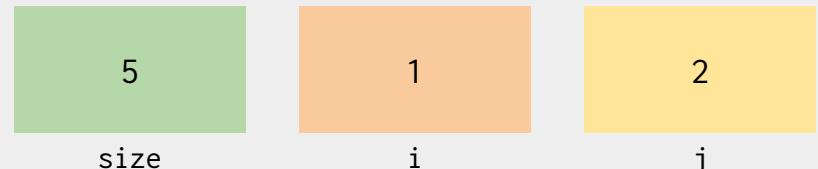


Walkthrough

```
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];
```

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

oneD



1
min

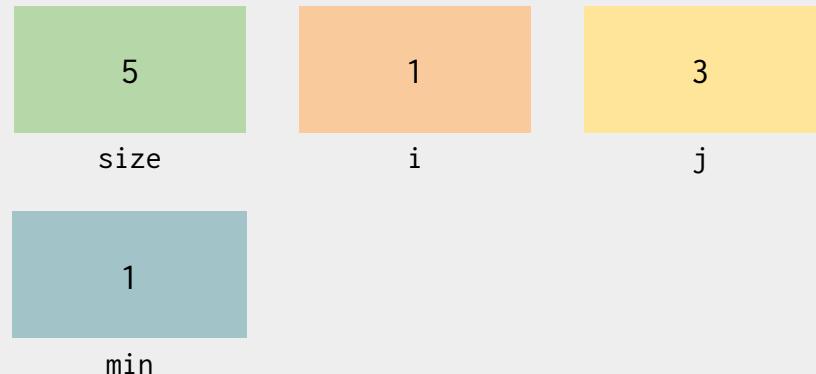


Walkthrough

```
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];
```

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

oneD

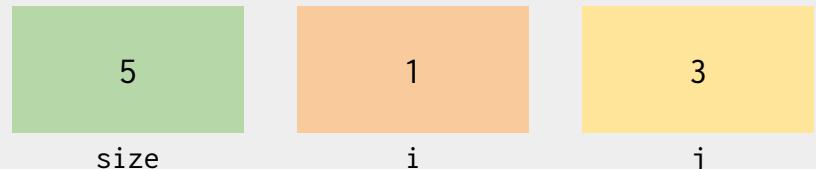


Walkthrough

```
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];
```

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

oneD

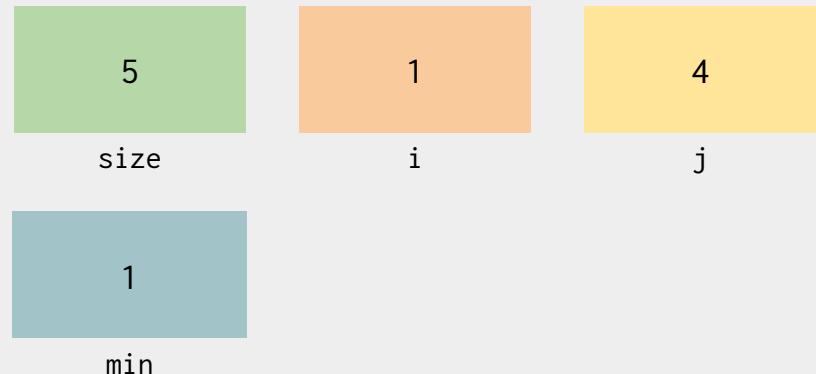


Walkthrough

```
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];
```

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

oneD

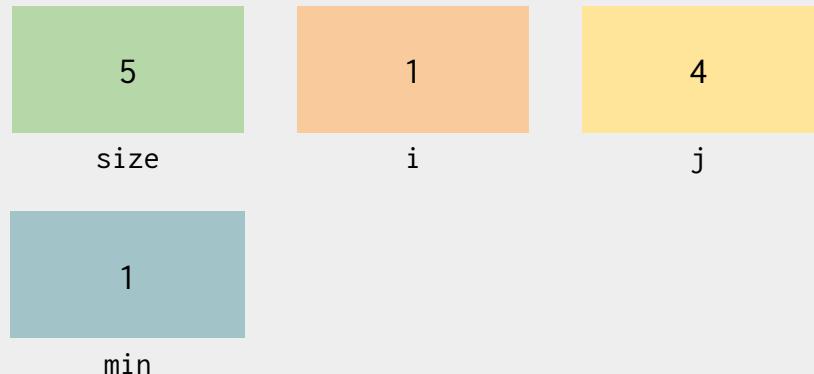


Walkthrough

```
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];
```

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

oneD

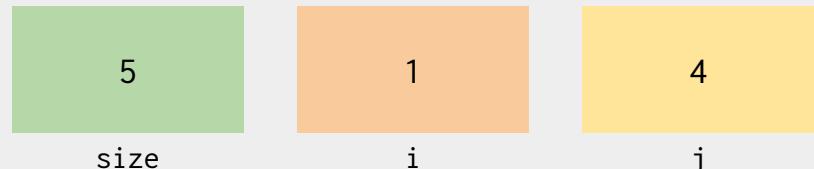


Walkthrough

```
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];
```

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

oneD



4

min

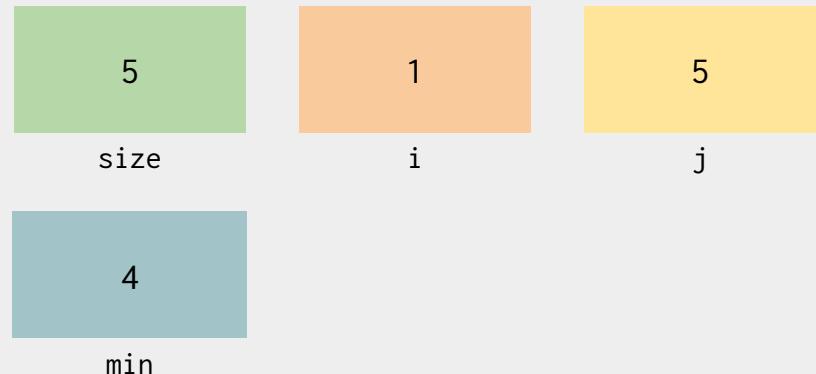


Walkthrough

```
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];
```

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

oneD

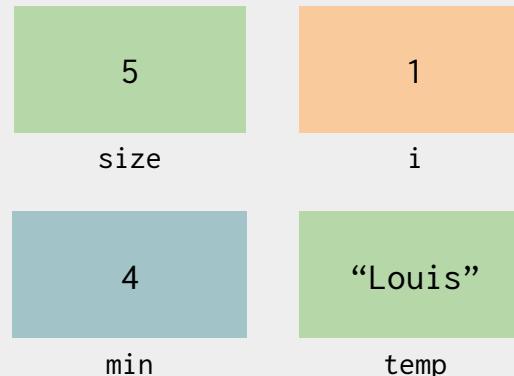


Walkthrough

```
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];
```

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

oneD

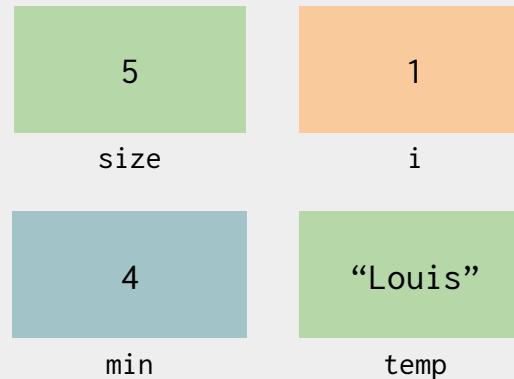


Walkthrough

```
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];
```

["Harry", "Liam", "Zayn", "Niall", "Liam"]

oneD

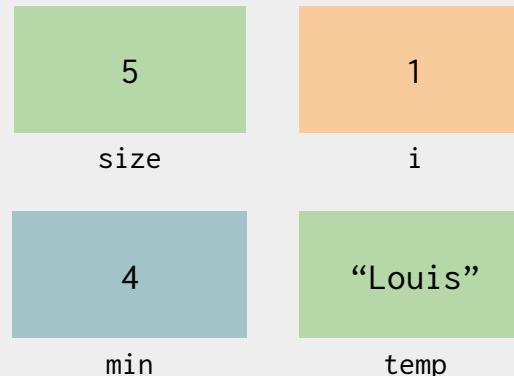


Walkthrough

```
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];
```

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

oneD



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: Transpose

Implement a function that takes in a pointer to an $n \times n$ 2d array of ints and transposes it. That is, the rows should become the columns and vice versa.

```
void transpose(int** matrix, int n);
```

Example: 1 2 3 ($n = 3$) \rightarrow 1 4 7

4 5 6	2 5 8
7 8 9	3 6 9



Solution: Transpose

```
void transpose(int** matrix, int n) {  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < i; j++) {  
            int temp = matrix[i][j];  
            matrix[i][j] = matrix[j][i];  
            matrix[j][i] = temp;  
        }  
    }  
}
```



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

x



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’, ‘s’, …]
```

x



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’, ‘s’, ‘\0’, ...]
```

x



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.1b-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];
}
```



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  
}
```

When foo called →

120	string s
116	int k
100	int a[4]
0-100	Variables in the calling function.

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 foo returns

120	Vacant
116	Vacant
100	Vacant
0-100	Variables in the calling function.

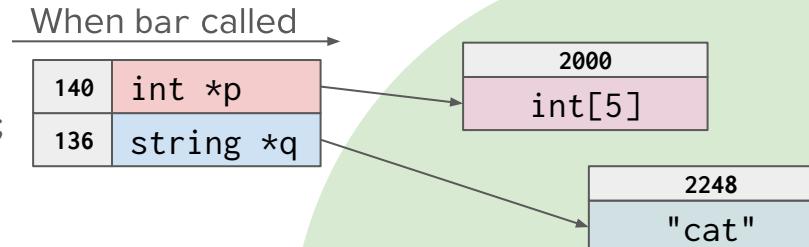
When the function returns, the variables are evicted from their addresses.



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");  
}
```

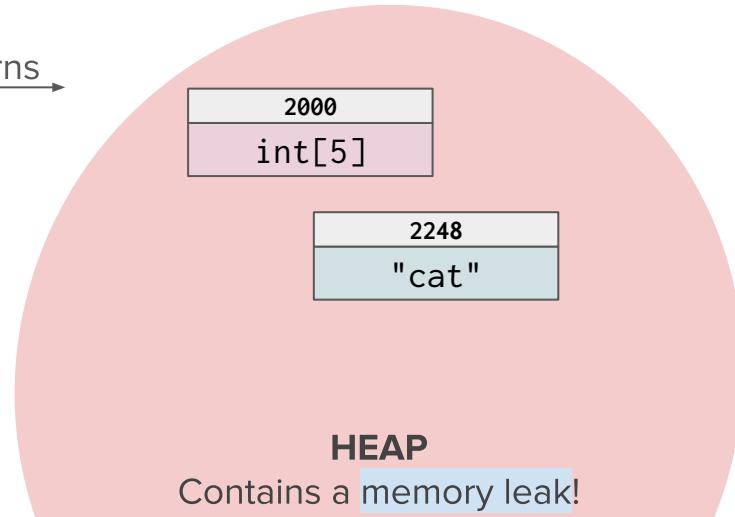


Pointers – the Heap and the Stack (cont.)

- 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");  
}
```

When bar returns

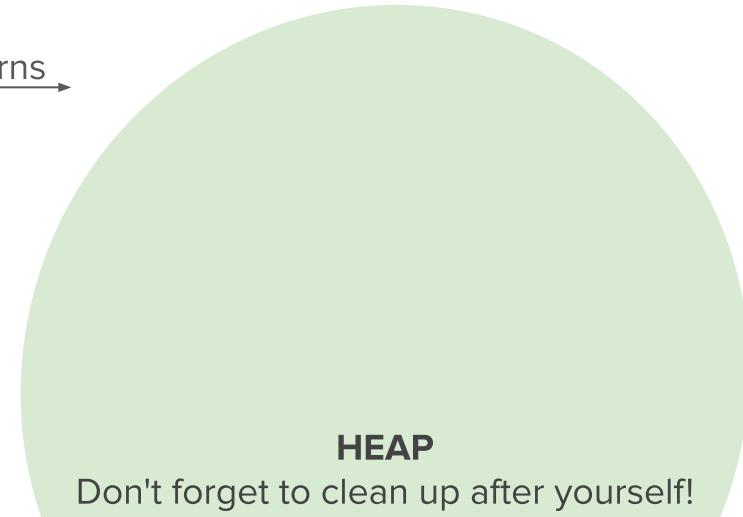


Pointers – the Heap and the Stack (cont.)

- 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;  
}
```

When bar returns →



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; }
```



Structs

- A **struct** is a collection of data that is treated as its own special data type. We use them to organize data that belongs together.

```
struct Person {  
    int age;  
    string name;  
}; // Must end with semicolon!
```

- Structs can store any number of any other data type, accessed using . (the **dot operator**). These stored values are called **member variables**.



Structs

You can declare a struct outside of any functions and treat it as a normal variable, for the most part. A struct can even contain another struct!

```
struct Date {  
    int day, month, year;  
};  
  
struct Person {  
    Date birthday;  
    string name;  
    double money;  
};
```

```
void doubleMoney(Person& guy) {  
    guy.money *= 2;  
}  
  
int main() {  
    Person p1;  
    p1.name = "Smelborp";  
    p1.money = 3.50;  
    p1.birthday.day = p1.birthday.month = 1;  
    Person p2 = p1; // Perfectly legal  
    doubleMoney(p2);  
    cout << p2.money; // 7  
    cout << p1.money; // 3.5  
    Person p3 = { p1.birthday, "Jimbo", 3.5 };  
    p2 += p1; // ERROR! How do you add people?!
```



Structs – Tips

- Structs are a good way to keep code looking organized and readable
- When declaring a struct, member variables with primitive types will be left uninitialized. Classes will be constructed with their default constructor.
 - Long story short: you must assign them yourself!
- **Always remember the semicolon!** It's there so that you can declare struct variables at the same time that you define the struct.

```
struct Circle {  
    int radius;  
} ring, hoop;    // This creates two Circle structs named ring and hoop
```



Good luck!

Sign-in <http://bit.ly/2QCmwRH>

Slides <http://bit.ly/2NXAYIs>

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.

