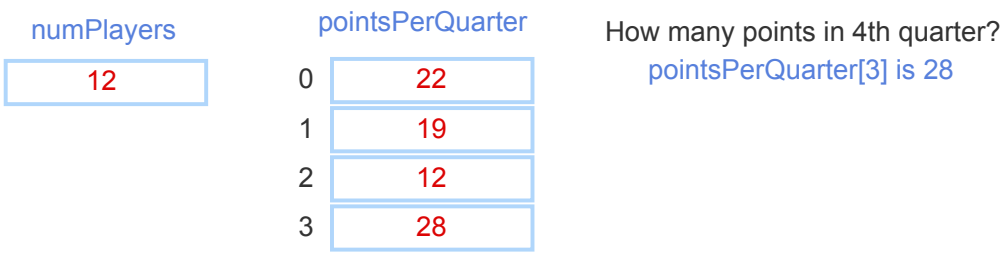
5.1 Array/vector concept (general)

A typical variable stores one data item, like the number 59 or the character 'a'. Instead, sometimes a *list* of data items should be stored. Ex: A program recording points scored in each quarter of a basketball game needs a list of 4 numbers. Requiring a programmer to declare 4 variables is annoying; 200 variables would be ridiculous. An ***array*** is a special variable having one name, but storing a list of data items, with each item being directly accessible. Some languages use a construct similar to an array called a ***vector***. Each item in an array is known as an ***element***.

PARTICIPATION ACTIVITY 5.1.1: Sometimes a variable should store a list, or array, of data items.



Animation content:

Static Figure: Variable numPlayers stores the number 12 and the variable pointsPerQuarter is an array that stores 4 items.

Step 1: A variable usually stores just one data item.

Step 2: Some variables should store a list of data items, like variable pointsPerQuarter that stores 4 items. Elements 22, 19, 12, and 28 are stored in pointsPerQuarter. Element 22 is stored at location number 0, element 19 is stored at location number 1, element 12 is stored at location number 2, and element 28 is stored at location number 3.

Step 3: Each element is accessible, like the element numbered 3. The answer to the question "How many points in 4th quarter?" is 28 because element 28 is the fourth item in pointsPerQuarter and is located at location number 3.

Animation captions:

1. A variable usually stores just one data item.
2. Some variables should store a list of data items, like variable pointsPerQuarter that stores 4 items.

3. Each element is accessible, like the element numbered 3.

You might think of a normal variable as a truck, and an array variable as a train. A truck has just one car for carrying "data", but a train has many cars, each of which can carry data.

Figure 5.1.1: A normal variable is like a truck, whereas an array variable is like a train.

Source: Truck (Ian Britton / freefoto.com), Train (Ian Britton / freefoto.com)

In an array, each element's location number is called the ***index***, myArray[2] has index 2. An array's key feature is that the index enables direct access to any element, as in myArray[2]; different languages may use different syntax, like myArray(3) or myVector.at(3). In many languages, indices start with 0 rather than 1, so an array with 4 elements has indices 0, 1, 2, and 3.

PARTICIPATION ACTIVITY 5.1.2: Update the array's data values.

Start

Update myItems with the given code.

myItems

0	63
1	18
2	80
3	12
4	39
5	41
6	41

1 2 3 4 5 6

Check Next

PARTICIPATION ACTIVITY 5.1.3: Array basics.

Array peoplePerDay has 365 elements, one for each day of the year. Valid accesses are peoplePerDay[0], [1], ..., [364].

1) Which assigns element 0 with the value 250?

- peoplePerDay[250] = 0
- peoplePerDay[0] = 250
- peoplePerDay = 250

2) Which assigns element 1 with the value 99?

- peoplePerDay[1] = 99
- peoplePerDay[99] = 1

3) What is the value of peoplePerDay[8]?

```
peoplePerDay[9] = 5;
peoplePerDay[8] =
peoplePerDay[9] - 3;
```

- 8
- 5
- 2

4) Assume N is initially 1. What is the value of peoplePerDay[2]?

```
peoplePerDay[N] = 15;
N = N + 1;
peoplePerDay[N] = peoplePerDay[N
- 1] * 3;
```


- 15
- 2
- 45

PARTICIPATION ACTIVITY 5.1.4: Arrays with element numbering starting with 0.

Array scoresList has 10 elements with indices 0 to 9, accessed as scoresList[0] to scoresList[9].

1) Assign the first element in scoresList with 77.

Check Show answer

2) Assign the second element in scoresList with 77.

Check Show answer

3) Assign the last element with 77.

Check Show answer

4) If that array instead has 100 elements, what is the last element's index?

Check Show answer

5) If the array's last index was 499, how many elements does the array have?

Check Show answer

5.2 Arrays

Array declarations and accessing elements

A programmer commonly needs to maintain a list of items, just as people often maintain lists of items like a grocery list or a course roster. An ***array*** is an ordered list of items of a given data type. Each item in an array is called an ***element***.

Construct 5.2.1: Array declaration.

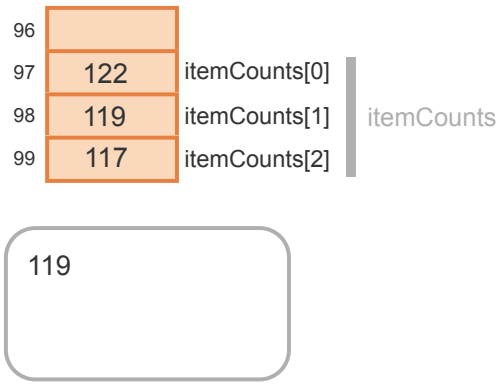

```
dataType
arrayName[numElements];
```

This statement declares an array having the specified number of elements in memory, each element of the specified data type. The desired number of elements are specified in [ ] symbols.

Terminology note: [ ] are ***brackets***. { } are ***braces***. In an array access, the number in brackets is called the ***index*** of the corresponding element. The first array element is at index 0.

PARTICIPATION ACTIVITY 5.2.1: An array declaration creates multiple variables in memory, each accessible using [ ].

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

int main() {
   int itemCounts[3];

   itemCounts[0] = 122;
   itemCounts[1] = 119;
   itemCounts[2] = 117;

   cout << itemCounts[1];

   return 0;
}
```



Animation content:

Static figure:
Begin C++ code:
#include <iostream>
using namespace std;

int main() {
  int itemCounts[3];

```
   itemCounts[0] = 122;
   itemCounts[1] = 119;
   itemCounts[2] = 117;

   cout << itemCounts[1];

   return 0;
}
```

End C++ code.

There is a memory box with four segmented rows contained inside, all of which are empty. The rows are numbered from top to bottom: 96, 97, 98, and 99. There is another box which displays the output, and is also empty.

Step1: An array named itemCounts is declared. The array consists of 3 elements, each of data type int. Rows 97, 98, and 99 are all labeled "itemCounts". Row 97 is labeled "itemCounts[0]". Row 98 is labeled "itemCounts[1]". Row 99 is labeled "itemCounts[2]".

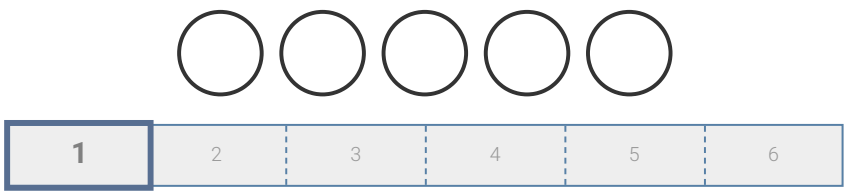
Step 2: An element is accessed with brackets. The number in brackets is called the index of the corresponding element. The line of code, "itemCounts[0] = 122;", is highlighted and 122 is stored into row 97 of the memory box. The line of code, "itemCounts[1] = 119;", is highlighted and 119 is stored into row 98 of the memory box. The line of code, "itemCounts[2] = 117;", is highlighted and 117 is stored into row 99 of the memory box. The line of code "cout << itemCounts[1];" is highlighted, row 98 of the memory box is accessed, and the stored integer (119) is output. Thus, the output display contains the text, "119."

Animation captions:

1. An array named itemCounts is declared. The array consists of 3 elements, each of data type int.
2. An element is accessed with brackets. The number in brackets is called the index of the corresponding element.

PARTICIPATION ACTIVITY 5.2.2: Select the index shown.

Start



Check Next

PARTICIPATION ACTIVITY 5.2.3: Array basics.

Given:


```
int yearsArr[4];

yearsArr[0] = 1999;
yearsArr[1] = 2012;
yearsArr[2] = 2025;
```

1) How many elements in memory does the array declaration create?

- 0
- 1
- 3
- 4

2) What value is stored in yearsArr[1]?

- 1
- 1999
- 2012

3) With what value does `currYear = yearsArr[2]` assign currYear?

- 2
- 2025
- Invalid index

4) With what value does `currYear = yearsArr[3]` assign currYear?

- 3
- 2025
- Invalid index
- Unknown

5) Recall that the array declaration was `int yearsArr[4]`. Is `currYear = yearsArr[4]` a valid assignment?

- Yes, it accesses the fourth element.
- No, yearsArr[4] does not exist.

6) What is the proper way to access the *first* element in array yearsArr?

- yearsArr[1]
- yearsArr[0]

7) What are the contents of the array if the above code is followed by the statement: yearsArr[0] = yearsArr[2]?

- 1999, 2012, 1999, ?
- 2012, 2012, 2025, ?
- 2025, 2012, 2025, ?

8) What is the index of the *last* element for the following array: `int pricesArr[100];`

- 99
- 100
- 101

Using an expression for an array index

A powerful aspect of arrays is that the index is an expression. Ex: userNums[i] uses the value held in the int variable `i` as the index. As such, an array is useful to easily lookup the Nth item in a list.

An array's index must be an integer type. The array index cannot be a floating-point type, even if the value is 0.0, 1.0, etc.

The program below allows a user to print the age of the Nth oldest known person to have ever lived. The program quickly accesses the Nth oldest person's age using `oldestPeople[nthPerson - 1]`. Note that the index is `nthPerson - 1` rather than just `nthPerson` because an array's indices start at 0, so the 1st age is at index 0, the 2nd at index 1, etc.

Figure 5.2.1: Array's nth element can be directly accessed using [n-1]: Oldest people program.

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

int main() {
   int oldestPeople[5];
   int nthPerson;       // User input, Nth oldest
person

   oldestPeople[0] = 122; // Died 1997 in France
   oldestPeople[1] = 119; // Died 1999 in U.S.
   oldestPeople[2] = 117; // Died 1993 in U.S.
   oldestPeople[3] = 117; // Died 1998 in Canada
   oldestPeople[4] = 116; // Died 2006 in Ecuador

   cout << "Enter N (1..5): ";
   cin  >> nthPerson;

   if ((nthPerson >= 1) && (nthPerson <= 5)) {
      cout << "The " << nthPerson << "th oldest
person lived ";
      cout << oldestPeople[nthPerson - 1] << "
years." << endl;
   }

   return 0;
}
```

```
Enter N (1..5): 1
The 1th oldest person lived
122 years.

...
Enter N (1..5): 4
The 4th oldest person lived
117 years.

...

Enter N (1..5): 9

...

Enter N (1..5): 0

...

Enter N (1..5): 5
The 5th oldest person lived
116 years.
```


PARTICIPATION ACTIVITY 5.2.4: Nth oldest person program.

1) In the program above, what is the purpose of this if statement:

```
if ((nthPerson >= 1) &&
(nthPerson <= 5)) {
   ...
}
```

- To avoid overflow because nthPerson's data type can only store values from 1 to 5.
- To ensure only valid array elements are accessed because the array oldestPeople only has 5 elements.

PARTICIPATION ACTIVITY 5.2.5: Array declaration and accesses.

1) Declare an array named myVals that stores 10 items of type int.

Check Show answer

2) Assign variable x with the value stored at index 8 of array myVals.

Check Show answer

3) Assign the second element of array myVals with the value 555.

Check Show answer

4) Assign myVals array element at the index held in currIndex with the value 777.

Check Show answer

5) Assign tempVal with the myVals array element at the index one after the value held in variable i.

Check Show answer

Loops and arrays

A key advantage of arrays becomes evident when used in conjunction with loops. The program below uses a loop to allow a user to enter 8 integer values, storing those values in an array, and then printing those 8 values.

Figure 5.2.2: Arrays combined with loops are powerful together: User-entered numbers.

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

int main() {
   const int NUM_ELEMENTS = 8;  // Number of
elements in array
   int userVals[NUM_ELEMENTS];  // User numbers
   int i;                       // Loop index

   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   cout << "You entered: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << " ";
   }
   cout << endl;

   return 0;
}
```



```
Enter 8 integer values...
Value: 5
Value: 99
Value: -1
Value: -44
Value: 8
Value: 555555
Value: 0
Value: 2
You entered: 5 99 -1 -44 8
555555 0 2
```

PARTICIPATION ACTIVITY 5.2.6: Array with loops.

Refer to the program above.

1) How many times does each for loop iterate?

   - 1
   - 8
   - Unknown

2) Which one line of code can be changed to allow the user to enter 100 elements?

   - `const int NUM_ELEMENTS = 8;`
   - `for (i = 0; i < NUM_ELEMENTS; ++i) {`

Array initialization

An array's elements are not automatically initialized during the variable declaration and should be initialized before being read. A programmer may initialize an array's elements in an array variable declaration by specifying the initial values in braces {} separated by commas. Ex: `int myArray[3] = {5, 7, 11};` initializes element at index 0 with 5, element at index 1 with 7, and element at index 2 with 11. For larger arrays, a loop may be used for initialization.

Like other variables, the keyword ***const*** may be prepended to an array variable declaration to prevent changes to the array. Thus, `const int YEARS[3] = {1865, 1920, 1964};` followed by `YEARS[0] = 2000;` yields a compiler error.

PARTICIPATION ACTIVITY 5.2.7: Array initialization.

1) Declare an array of ints named myVals with 4 elements each initialized to 10. The array declaration and initialization should be done in a single statement.

Check Show answer

2) Given: int maxScores[4] = {20, 20, 100, 50};. What is maxScores[3]?

Check Show answer

CHALLENGE ACTIVITY 5.2.1: Enter the output for the array.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   const int NUM_ELEMENTS = 3;
   int userVals[NUM_ELEMENTS];
   int i;

   userVals[0] = 1;
   userVals[1] = 6;
   userVals[2] = 9;

   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << endl;
   }

   return 0;
}
```

```
1
6
9
```


1 2 3 4

Check Next

CHALLENGE ACTIVITY 5.2.2: Printing array elements.

Write three statements to print the first three elements of array runTimes. Follow each statement with a newline. Ex: If runTimes = {800, 775, 790, 805, 808}, print:

```
800
775
790
```

Note: These activities will test the code with different test values. This activity will perform two tests, both with a 5-element array (int runTimes[5]). See "How to Use zyBooks".

Also note: If the submitted code tries to access an invalid array element, such as runTime[9] for a 5-element array, the test may generate strange results. Or the test may crash and report "Program end never reached", in which case the system doesn't print the test case that caused the reported message.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5    const int NUM_ELEMENTS = 5;
6    int runTimes[NUM_ELEMENTS];
7    int i;
```

```
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cin >> runTimes[i];
   }

   /* Your solution goes here  */

   return 0;
}
```


Run

CHALLENGE ACTIVITY 5.2.3: Printing array elements with a for loop.

Write a for loop to print all elements in courseGrades, following each element with a space (including the last). Print forwards, then backwards. End each loop with a newline. Ex: If courseGrades = {7, 9, 11, 10}, print:

```
7 9 11 10
10 11 9 7
```

Hint: Use two for loops. Second loop starts with i = NUM_VALS - 1. (Notes)

Note: These activities may test code with different test values. This activity will perform two tests, both with a 4-element array (int courseGrades[4]). See "How to Use zyBooks".

Also note: If the submitted code tries to access an invalid array element, such as courseGrades[9] for a 4-element array, the test may generate strange results. Or the test may crash and report "Program end never reached", in which case the system doesn't print the test case that caused the reported message.

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   const int NUM_VALS = 4;
   int courseGrades[NUM_VALS];
   int i;

   for (i = 0; i < NUM_VALS; ++i) {
      cin >> courseGrades[i];
   }

```

```
13    /* Your solution goes here  */
14
15    return 0;
```

Run

5.3 Iterating through arrays

Iterating through an array using loops

Iterating through arrays using loops is commonplace and is an important programming skill to master.

Because array indices are numbered 0 to N - 1 rather than 1 to N, programmers commonly use this for loop structure:

Figure 5.3.1: Common for loop structure for iterating through an array.

```
// Iterating through myArray
for (i = 0; i < numElements; ++i)
{
   // Loop body accessing
myArray[i]
}
```

Note that index variable i is initialized to 0, and the loop expression is i < N rather than i <= N. If N were 5, the loop's iterations would set i to 0, 1, 2, 3, and 4, for a total of 5 iterations. The benefit of the loop structure is that each array element is accessed as myArray[i] rather than the more complex myArray[i - 1].

PARTICIPATION ACTIVITY 5.3.1: Iterating through an array.

1) Complete the code to print all items for the given array, using the above common loop structure.

```
int daysList[365];
...

for (i = 0;
                    ; ++i) {
   cout << daysList[i] <<
endl;
}
```

Check Show answer

2) Given that this loop iterates over all items of the array, how many items are in the array?

```
for (i = 0; i < 99; ++i) {
   cout << someArray[i] <<
endl;
}
```

Check Show answer

Determining a quantity about an array's items

Iterating through an array for various purposes is an important programming skill to master. The program below computes the sum of an array's element values. For computing the sum, the program initializes a variable sum to 0, then simply adds the current iteration's array element value to that sum.

Figure 5.3.2: Iterating through an array example: Program that computes the sum of an array's elements.

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

int main() {
   const int NUM_ELEMENTS = 8; // Number of elements
   int userVals[NUM_ELEMENTS]; // User numbers
   int i;                      // Loop index
   int sumVal;                 // For computing sum

   // Prompt user to populate array
   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;

   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   // Determine sum
   sumVal = 0;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      sumVal = sumVal + userVals[i];
   }

   cout << "Sum: " << sumVal << endl;

   return 0;
}
```

```
Enter 8 integer
values...
Value: 3
Value: 5
Value: 234
Value: 346
Value: 234
Value: 73
Value: 26
Value: -1
Sum: 920

...

Enter 8 integer
values...
Value: 3
Value: 5
Value: 234
Value: 346
Value: 234
Value: 73
Value: 26
Value: 1
Sum: 922
```


Finding the maximum value in an array

Programs commonly iterate through arrays to determine some quantity about the array's items. The program below determines the maximum value in a user-entered list. If the user enters numbers 7, -9, 55, 44, 20, -400, 0, 2, then the program will output "max: 55". The program uses the variable maxVal to store the largest value seen "so far" as the program iterates through the array. During each iteration, if the array's current element value is larger than the max seen so far, the program assigns maxVal with the array element.

Before entering the loop, maxVal must be initialized to some value because max will be compared with each array element's value. A logical error would be to initialize maxVal to 0, because 0 is not in fact the largest value seen so far, and would result in incorrect output (of 0) if the user entered all negative numbers. Instead, the program peeks at an array element (using the first element, though any element could have been used) and initializes maxVal to that element's value.

Figure 5.3.3: Iterating through an array example: Program that finds the max item.

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

int main() {
   const int NUM_ELEMENTS = 8;  // Number of elements
   int userVals[NUM_ELEMENTS];  // Array of user numbers
   int i;                       // Loop index
   int maxVal;                  // Computed max

   // Prompt user to populate array
   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;

   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   // Determine largest (max) number
   maxVal = userVals[0];        // Largest so far

   for (i = 0; i < NUM_ELEMENTS; ++i) {
      if (userVals[i] > maxVal) {
         maxVal = userVals[i];
      }
   }
   cout << "Max: " << maxVal << endl;

   return 0;
}
```

```
Enter 8 integer
values...
Value: 3
Value: 5
Value: 23
Value: -1
Value: 456
Value: 1
Value: 6
Value: 83
Max: 456

...

Enter 8 integer
values...
Value: -5
Value: -10
Value: -44
Value: -2
Value: -27
Value: -9
Value: -27
Value: -9
Max: -2
```


PARTICIPATION ACTIVITY 5.3.2: Array iteration.

Given an integer array myVals of size N_SIZE (i.e. int myVals[N_SIZE]), complete the code to achieve the stated goal.

1) Determine the minimum number in the array, using the same initialization as the maximum number example above.

```
minVal = [          ];
for (i = 0; i < N_SIZE; ++i) {
   if (myVals[i] < minVal) {
      minVal = myVals[i];
   }
}
```

Check Show answer

2) Count how many negative numbers exist in the array.

```
cntNeg = 0;
for (i = 0; i < N_SIZE; ++i) {
   if (                ) {
      ++cntNeg;
   }
}
```

Check Show answer

3) Count how many odd numbers exist in the array.

```
cntOdd = 0;
for (i = 0; i < N_SIZE; ++i) {
   if ( (myVals[i] % 2) == 1 )
{
          ;
   }
}
```

Check Show answer

zyDE 5.3.1: Print the sum and average of an array's elements.

Modify the program to print the average (mean) as well as the sum. Hint: You don't actually have to change the loop, but rather change what you print.

Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     const int NUM_ELEMENT
6     int userVals[NUM_ELEM
7     int i;
8     int sumVal;
9
10    // Prompt user to pop
11    cout << "Enter " << N
12
13    for (i = 0; i < NUM_E
14       cout << "Value: "
15       cin >> userVals[i
```

3 5 234 346 234 73 26 -1

Run

zyDE 5.3.2: Print selected elements of an array.

Modify the program to instead just print each number that is greater than 21.

Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     const int NUM_ELEMENT
6     int userVals[NUM_ELEM
7     int i;
8     int sumVal;
9
10    // Prompt user to pop
11    cout << "Enter " << N
12
13    for (i = 0; i < NUM_E
14       cout << "Value: "
15       cin >> userVals[i
```

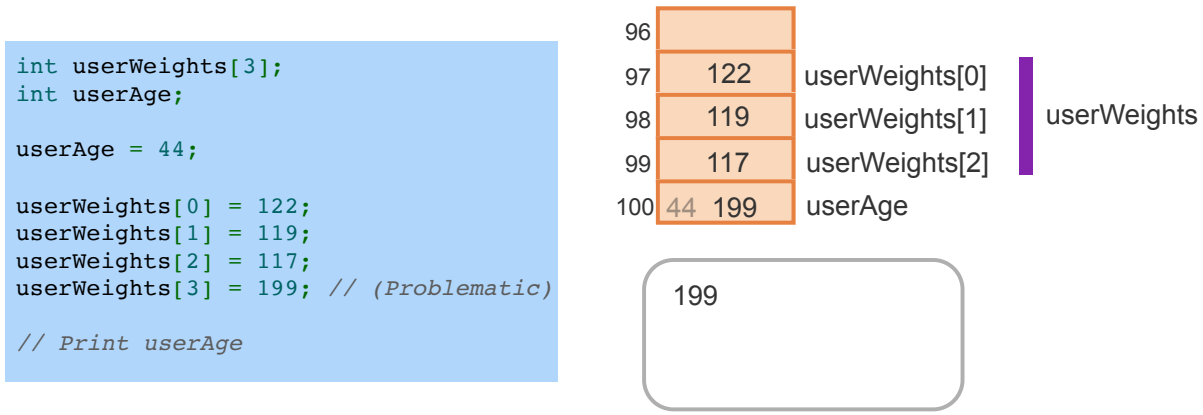
3 5 234 346 234 73 26 -1

Run

Common error: Accessing out of range array element

A common error is to try to access an array with an index that is out of the array's index range. Ex: Trying to access highScores[8] when highScores's valid indices are 0-7. Care should be taken whenever a user enters a number that is then used as an array index, and when using a loop index as an array index also, to ensure the index is within the array's valid index range. Checking whether an array index is in range is very important. Trying to access an array with an out-of-range index is not only a very common error, but is also one of the hardest errors to debug. The following animation shows what happens when a program writes to an out-of-range index using an array.

PARTICIPATION ACTIVITY 5.3.3: Writing to an out-of-range index using an array.



Animation content:

Static figure:
Begin code.
int userWeights[3];
int userAge;

userAge = 44;

userWeights[0] = 122;
userWeights[1] = 119;
userWeights[2] = 117;
userWeights[3] = 199; *// (Problematic)*

// Print userAge
End code.

There is a memory box with five segmented rows contained inside, all of which are empty. The rows are numbered from top to bottom: 96, 97, 98, 99, and 100. Rows 97, 98, and 99 are all labeled "userWeights". Row 97 is labeled "userWeights[0]". Row 98 is labeled "userWeights[1]". Row 99 is

labeled "userWeights[2]". Row 100 is labeled "userAge". There is another box which displays the output, and is also empty.

Step 1: Variable userAge is allocated a location in memory immediately after the array userWeights. userAge is assigned with 44. The line of code, "userAge = 44;", is highlighted and 44 is stored into row 100 of the memory box.

Step 2: Each element of array userWeights is assigned a value. The line of code, "userWeights[0] = 122;", is highlighted and 122 is stored into row 97 of the memory box. The line of code, "userWeights[1] = 119;", is highlighted and 119 is stored into row 98 of the memory box. The line of code, "userWeights[2] = 117;", is highlighted and 117 is stored into row 99 of the memory box.

Step 3: Index 3 is out of userWeights's index range, and results in overwriting userAge's value. The line of code, "userWeights[3] = 199; // (Problematic)", is highlighted and 199 is stored into row 100 of the memory box, replacing the previously stored value, 44. The line of code "// Print userAge" is highlighted, row 100 of the memory box is accessed, and the stored integer (199) is output. Thus, the output display contains the text, "199."

Animation captions:

1. Variable userAge is allocated a location in memory immediately after the array userWeights. userAge is assigned with 44.
2. Each element of array userWeights is assigned a value.
3. 3 is out of userWeights's index range, and results in overwriting userAge's value.

A write to an array with an out-of-range index may simply write to a memory location of a different variable X residing next to the array in memory. Later, when the program tries to read X, the program encounters incorrect data. For example, a program may write X with the number 44, but when reading X later in the program X may be 2533, with X never (intentionally) written by any program statement in between.

PARTICIPATION ACTIVITY 5.3.4: Iterating through an array.

Given the following code:

```
const int NUM_ELEMENTS = 5;
int myArray[NUM_ELEMENTS];
int i;
```


1) The normal for loop structure iterates as long as:

   `i <= NUM_ELEMENTS`

   - True
   - False

2) To compute the sum of elements, a reasonable statement preceding the for loop is: `sumVal = 0;`

- True
- False

3) To find the maximum element value, a reasonable statement preceding the for loop is: `maxVal = 0;`

- True
- False

CHALLENGE ACTIVITY 5.3.1: Enter the output for the array.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int main() {
   const int NUM_ELEMENTS = 3;
   int userVals[NUM_ELEMENTS];
   int i;

   userVals[0] = 1;
   userVals[1] = 4;
   userVals[2] = 7;

   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << endl;
   }

   return 0;
}
```

```
1
4
7
```

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 5.3.2: Finding values in arrays.

Assign numMatches with the number of elements in userValues that equal matchValue. userValues has NUM_VALS elements. Ex: If userValues is {2, 1, 2, 2} and matchValue is 2 ,

then numMatches should be 3.

Your code will be tested with the following values:
matchValue: 2, userValues: {2, 1, 2, 2} (as in the example program above)
matchValue: 0, userValues: {0, 0, 0, 0}
matchValue: 10, userValues: {20, 50, 70, 100}
(Notes)

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   const int NUM_VALS = 4;
   int userValues[NUM_VALS];
   int i;
   int matchValue;
   int numMatches = -99; // Assign numMatches with 0 before your for loop

   cin >> matchValue;
   for (i = 0; i < NUM_VALS; ++i) {
      cin >> userValues[i];
   }

```

Run

CHALLENGE ACTIVITY 5.3.3: Populating an array with a for loop.

Write a for loop to populate array userGuesses with NUM_GUESSES integers. Read integers using cin. Ex: If NUM_GUESSES is 3 and user enters 9 5 2, then userGuesses is {9, 5, 2}.

Learn how our autograder works

539740.3879454.qx3zqy7


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

int main() {
   const int NUM_GUESSES = 3;
   int userGuesses[NUM_GUESSES];
   int i;

   /* Your solution goes here  */

```

```
   for (i = 0; i < NUM_GUESSES; ++i) {
      cout << userGuesses[i] << " ";
   }

   return 0;
```

Run

CHALLENGE ACTIVITY 5.3.4: Array iteration: Sum of excess.

Array testGrades contains NUM_VALS test scores. Write a for loop that sets sumExtra to the total extra credit received. Full credit is 100, so anything over 100 is extra credit. Ex: If testGrades = {101, 83, 107, 90}, then sumExtra = 8, because 1 + 0 + 7 + 0 is 8.

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   const int NUM_VALS = 4;
   int testGrades[NUM_VALS];
   int i;
   int sumExtra = -9999; // Assign sumExtra with 0 before your for loop

   for (i = 0; i < NUM_VALS; ++i) {
      cin >> testGrades[i];
   }

   /* Your solution goes here  */

```

Run

CHALLENGE ACTIVITY 5.3.5: Printing array elements separated by commas.

Write a for loop to print all NUM_VALS elements of array hourlyTemp. Separate elements with a comma and space. Ex: If hourlyTemp = {90, 92, 94, 95}, print:

```
90, 92, 94, 95
```

Your code's output should end with the last element, without a subsequent comma, space, or newline.

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   const int NUM_VALS = 4;
   int hourlyTemp[NUM_VALS];
   int i;

   for (i = 0; i < NUM_VALS; ++i) {
      cin >> hourlyTemp[i];
   }

   /* Your solution goes here  */

   cout << endl;
```

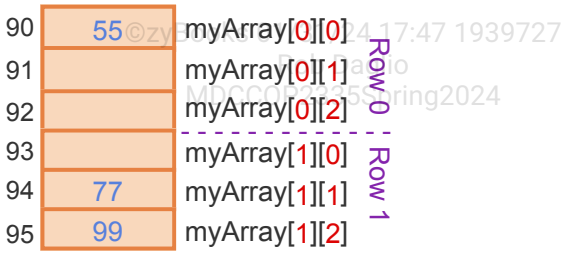

Run

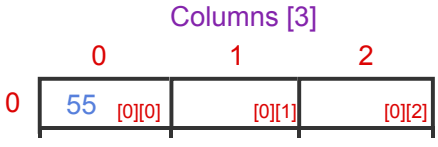
5.4 Two-dimensional arrays

An array can be declared with two dimensions. `int myArray[R][C]` represents a table of int variables with R rows and C columns, so R*C elements total. For example, `int myArray[2][3]` creates a table with 2 rows and 3 columns, for 6 int variables total. Example accesses are `myArray[0][0] = 33;` or `num = myArray[1][2]`.

PARTICIPATION ACTIVITY 5.4.1: Two-dimensional array.

```
// Define array with size [2][3]

// Write to some elements
myArray[0][0] = 55;
myArray[1][1] = 77;
myArray[1][2] = 99;
```





Animation content:

Begin static figure:
Begin C++ code:
// Define array with size [2][3]

// Write to some elements
myArray[0][0] = 55;
myArray[1][1] = 77;
myArray[1][2] = 99;
End C++ code.
A 2 by 3 matrix represents the two-dimensional myArray. Position [0][0] is 55. Position[1][1] is 77. Position [1][2] is 99. All other positions in the matrix are empty. In memory, row 0 is represented by memory positions 90 through 92. Position 90 is set to 55. Row 1 is represented by memory positions 93 through 95. Position 94 is set to 77, and position 95 is set to 99.

Step 1: Conceptually, a two-dimensional array is a table with rows and columns.
A 2 by 3 matrix is shown to represent the two-dimensional myArray. Position [0][0] is 55. Position[1][1] is 77. Position [1][2] is 99. All other positions in the matrix are empty.
Step 3: A two-dimensional array is implemented in a one-dimensional memory by placing each row following the previous row.
In memory, row 0 is represented by memory positions 90 through 92. Position 90 represents myArray[0][0], position 91 represents myArray[0][1], and position 92 represents myArray[0][2]. Row 1 is represented by memory positions 93 through 95. Position 93 represents myArray[1][0], position 94 represents myArray[1][1], and position 95 represents myArray[1][2].
The line of code reading "myArray[0][0] = 55;" is highlighted, and position 90 in memory is set to 55.
The line of code reading "myArray[1][1] = 77;" is highlighted, and position 94 in memory is set to 77.
The line of code reading "myArray[1][2] = 99;" is highlighted, and position 95 in memory is set to 99.

Animation captions:

1. Conceptually, a two-dimensional array is a table with rows and columns.
2. A two-dimensional array is implemented in a one-dimensional memory by placing each row following the previous row.

Conceptually, a two-dimensional array is a table with rows and columns. The compiler maps two-dimensional array elements to one-dimensional memory, each row following the previous row, known as ***row-major order***.

Figure 5.4.1: Using a two-dimensional array: A driving distance between cities example.

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

/* Direct driving distances between cities, in miles
*/
/* 0: Boston  1: Chicago  2: Los Angeles */

int main() {
   int cityA;                  // Starting city
   int cityB;                  // Destination city
   int drivingDistances[3][3]; // Driving distances

   // Initialize distances array
   drivingDistances[0][0] = 0;
   drivingDistances[0][1] = 960;  // Boston-Chicago
   drivingDistances[0][2] = 2960; // Boston-Los
Angeles
   drivingDistances[1][0] = 960;  // Chicago-Boston
   drivingDistances[1][1] = 0;
   drivingDistances[1][2] = 2011; // Chicago-Los
Angeles
   drivingDistances[2][0] = 2960; // Los Angeles-
Boston
   drivingDistances[2][1] = 2011; // Los Angeles-
Chicago
   drivingDistances[2][2] = 0;

   cout << "0: Boston  1: Chicago  2: Los Angeles"
<< endl;

   cout << "Enter city pair (Ex: 1 2) -- ";
   cin >> cityA;
   cin >> cityB;

   if ((cityA >= 0) && (cityA <= 2) && (cityB >= 0)
&& (cityB <= 2)) {
      cout << "Distance: " <<
drivingDistances[cityA][cityB];
      cout << " miles." << endl;
   }

   return 0;
}
```



```
0: Boston  1: Chicago  2:
Los Angeles
Enter city pair (Ex: 1 2)
-- 1 2
Distance: 2011 miles.

...

0: Boston  1: Chicago  2:
Los Angeles
Enter city pair (Ex: 1 2)
-- 2 0
Distance: 2960 miles.

...

0: Boston  1: Chicago  2:
Los Angeles
Enter city pair (Ex: 1 2)
-- 1 1
Distance: 0 miles.
```


A programmer can initialize a two-dimensional array's elements during declaration using nested braces, as below. Multiple lines make the rows and columns more visible.

Construct 5.4.1: Initializing a two-dimensional array during declaration.

```
// Initializing a 2D array
int numVals[2][3] = { {22, 44, 66}, {97, 98, 99}
};

// Use multiple lines to make rows more visible
int numVals[2][3] = {
   {22, 44, 66}, // Row 0
   {97, 98, 99}  // Row 1
};
```


Arrays of three or more dimensions can also be declared, as in `int myArray[2][3][5]`, which declares a total of 2*3*5 or 30 elements. Note the rapid growth in size -- an array declared as `int myArray[100][100][5][3]` would have 100*100*5*3 or 150,000 elements. A programmer should make sure not to unnecessarily occupy available memory with a large array.

PARTICIPATION ACTIVITY 5.4.2: Two-dimensional arrays.

1) Declare a two dimensional array of integers named dataVals with 4 rows and 7 columns.

Check Show answer

2) How many total elements are in an array with 4 rows and 7 columns?

Check Show answer

3) How many elements are in the array declared as: char streetNames[20][50];

Check Show answer

4) Write a statement that assigns 99 into the fifth row, third column of array numVals. Note: the first row/column is at index 0, not 1.

Check Show answer

CHALLENGE ACTIVITY 5.4.1: Find 2D array max and min.

Find the maximum value and minimum value in milesTracker. Assign the maximum value to maxMiles, and the minimum value to minMiles.

Ex: If the input is:

```
-10 20 30 40
```

the output is:

```
Min miles: -10
Max miles: 40
```

(Notes)

Learn how our autograder works

539740.3879454.qx3zqy7

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

int main() {
   const int NUM_ROWS = 2;
   const int NUM_COLS = 2;
   int milesTracker[NUM_ROWS][NUM_COLS];
   int i;
   int j;
   int maxMiles = 0; // Assign with first element in milesTracker before lo
   int minMiles = 0; // Assign with first element in milesTracker before lo
   int value;

   for (i = 0; i < NUM_ROWS; i++){
      for (j = 0; j < NUM_COLS; j++){
```

Run

5.5 Multiple arrays

Programmers commonly use multiple same-sized arrays to store related lists. The program below maintains a list of letter weights in ounces, and another list indicating the corresponding postage cost for first class mail (usps.com).

The `if (userLetterWeight <= letterWeights[i])` statement compares the user-entered letter weight with the current element in the letterWeights array. If the entered weight is less than or equal to the current element in the letterWeights array, the program prints the element in postageCosts at the same index.

The loop's expression `(i < NUM_ELEMENTS) && (!foundWeight)` depends on the value of the variable foundWeight. This expression prevents the loop from iterating through the entire array once the correct letter weight has been found. Omitting the check for found from the loop expression would result in an incorrect output; the program would incorrectly print the postage cost for all letter weights greater than the user's letter weight.

Figure 5.5.1: Multiple array example: Letter postage cost program.

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

int main () {
   const int NUM_ELEMENTS = 14;
   // Weights in ounces
   double letterWeights[NUM_ELEMENTS] = {1.0, 2.0, 3.0, 3.5, 4.0, 5.0, 6.0,
                                         7.0, 8.0, 9.0, 10.0, 11.0, 12.0,
13.0};
   // Costs in cents (usps.com 2017)
   int postageCosts[NUM_ELEMENTS] = {49, 70, 91, 112, 161, 182, 203,
                                     224, 245, 266, 287, 308, 329, 350};
   double userLetterWeight;
   bool foundWeight;
   int i;

   // Prompt user to enter letter weight
   cout << "Enter letter weight (in ounces): ";
   cin >> userLetterWeight;

   // Postage costs is based on smallest letter weight greater than
   // or equal to mailing letter weight
   foundWeight = false;

   for (i = 0; (i < NUM_ELEMENTS) && (!foundWeight); ++i) {
      if (userLetterWeight <= letterWeights[i]) {
         foundWeight = true;
         cout << "Postage for USPS first class mail is ";
         cout << postageCosts[i] << " cents" << endl;
      }
   }

   if( !foundWeight ) {
      cout << "Letter is too heavy for USPS first class mail." << endl;
   }

   return 0;
}
```



```
Enter letter weight (in ounces): 3
Postage for USPS first class mail is 91 cents

...

Enter letter weight (in ounces): 9.5
Postage for USPS first class mail is 287 cents

...

Enter letter weight (in ounces): 15
Letter is too heavy for USPS first class mail.
```


PARTICIPATION ACTIVITY 5.5.1: Multiple arrays in the above postage cost program.

1) letterWeights[0] is 1, meaning the first element of letterWeights and postageCosts correspond to a weight of 1 ounce.

- True
- False

2) postageCosts[2] represents the cost for a weight of 2 ounces.

- True
- False

3) The program fails to provide a cost for a weight of 7.5.

- True
- False

zyDE 5.5.1: Postage calculation.

Improve the program by also outputting "The next higher weight is ___ with a cost of ___ cents".

Load default template...

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

int main () {
   const int NUM_ELEMENT
   // Weights in ounces
   double letterWeights[

   // Costs in cents (us
   int postageCosts[NUM_

   double userLetterWeig
   bool foundWeight;
   int i;
```

3

Run

PARTICIPATION ACTIVITY 5.5.2: Multiple arrays.

1) Using two separate statements, declare two related integer arrays named seatPosition and testScore (in that order) each with 130 elements.

Check Show answer

2) How many total elements are stored within the two arrays int familyAges[50] and double familyHeights[50]?

Check Show answer

CHALLENGE ACTIVITY 5.5.1: Multiple arrays.

539740.3879454.qx3zqy7

Start

Subtract each element in origList with the corresponding value in offsetAmount. Print each difference followed by a semicolon (no spaces).

Ex: If origList = {4, 5, 10, 12} and offsetAmount = {2, 4, 7, 3}, print:

```
2;1;3;9;
```

```
#include <iostream>
#include <string.h>
using namespace std;

int main() {
   const int NUM_VALS = 4;
   int origList[NUM_VALS];
   int offsetAmount[NUM_VALS];
   int i;

   cin >> origList[0];
   cin >> origList[1];
   cin >> origList[2];
   cin >> origList[3];

```

```
    cin >> offsetAmount[0];
    cin >> offsetAmount[1];
    cin >> offsetAmount[2];
    cin >> offsetAmount[3];

    /* Your code goes here */

    cout << endl;

    return 0;
}
```


1 2

Check Next

5.6 Swapping two variables (general)

Sometimes a program must swap values among two variables. ***Swapping*** two variables x and y means to assign y's value to x, and x's value to y. If x is 33 and y is 55, then after swapping x is 55 and y is 33.

A common method for swapping uses a temporary variable. A ***temporary variable*** is a variable used briefly to store a value. To understand the intuition of such temporary storage, consider a person holding a book in one hand and a phone in the other, wishing to swap the items. The person can temporarily place the phone on a table, move the book to the other hand, then pick up the phone.

PARTICIPATION ACTIVITY 5.6.1: Swap idea: Use a temporary location.

1. Put phone on table
2. Move book
3. Pick up phone

Table
(temporary place)

Animation content:

Static figure: A hand holding a book, a table, and another hand holding a phone are displayed. A set of three instructions are displayed as well. The first instruction is to put phone on table. The second instruction is to move book. The third instruction is to pick up phone.

Step 1: A swap between two hands requires a third, temporary place. The first instruction to put phone on table is executed by temporarily placing the phone on the table. The hand that was holding the phone is now empty. The second instruction to move book is executed by moving the book to the empty hand. The hand that was holding the phone is now holding the book and the hand that was holding the book is now empty. The third instruction pick up phone is executed by picking up the phone with the empty hand.

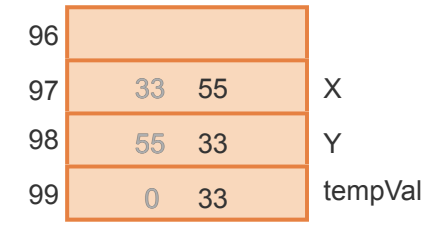
Animation captions:

1. A swap between two hands requires a third, temporary place.

Similarly, swapping two variables can use a third variable to temporarily hold one value while the other value is copied over.

PARTICIPATION ACTIVITY 5.6.2: Swapping two variables using a third temporary variable.

```
int X = 33;
int Y = 55;
int tempVal = 0;

tempVal = X;
X = Y;
Y = tempVal;

// Print X and Y
```



X: 55, Y: 33

Animation content:

Static figure: Variables X, Y, and tempVal are displayed, along with their respective original and updated values. X's original value is 33 and updated value is 55. Y's original value is 55 and updated

value is 33. tempVal's original value is 0 and updated value is 33. Both X and Y's updated values are output on a monitor.

Step 1: 55 is assigned to x. 33 is lost. The swap fails because no temporary variable was used.
Begin C++ code:

```
int X = 33;
int Y = 55;
int tempVal = 0;

X = Y;
Y = X;

// Print X and Y
```

End C++ code.
X is initialized with 33, Y is initialized with 55, and tempVal is initialized with 0. X is assigned with Y's value (55). Y is assigned with X's updated value (55). The text "X: 55, Y:55" is output on a monitor.

Step 2: The swap succeeds because x's value is saved before being assigned 55, and then 33 is assigned to y.
Begin C++ code:

```
int X = 33;
int Y = 55;
int tempVal = 0;

tempVal = X;
X = Y;
Y = tempVal;

// Print X and Y
```

End C++ code.
X is initialized with 33, Y is initialized with 55, and tempVal is initialized with 0. tempVal is assigned with X's value (33). X is assigned with Y's value (55). Y is assigned with tempVal's updated value (33). The text "X: 55, Y:33" is output on a monitor.

Animation captions:

1. 55 is assigned to x. 33 is lost. The swap fails because no temporary variable was used.
2. The swap succeeds because x's value is saved before being assigned 55, and then 33 is assigned to y.

PARTICIPATION ACTIVITY 5.6.3: Swap.

To begin, x is 22 and y is 99. What are x and y after the given code?

1)
```
x = y;
y = x;
```

- x is 99 and y is 22.
- x is 22 and y is 99.
- x is 99 and y is 99.

2)
```
x = y;
y = x;
x = y;
```


- x is 99 and y is 22.
- x is 99 and y is 99.
- x is 22 and y is 22.

3)
```
tempVal = x;
x = y;
y = x;
```

- x is 99 and y is 22.
- x is 99 and y is 99.

4)
```
tempVal = x;
x = y;
y = tempVal;
```

- x is 99 and y is 22.
- x is 99 and y is 99.

If you have studied arrays or vectors (or other kinds of lists), know that most swaps are actually performed between two list elements. For example, reversing a list with N elements can be achieved by swapping element 1 and N, element 2 and N-1, element 3 and N-2, etc. (stopping at the middle of the list).

PARTICIPATION ACTIVITY 5.6.4: Reversing a list using swaps.

Animation content:

Static figure: A list of integer elements are displayed. Integers 33, 55, 22, 77, and 11 are displayed, respectively.

Step 1: Swap outermost elements. The outermost elements (33 and 11) are swapped. The list is now 11, 55, 22, 77, and 33.

Step 2: Swap next outermost elements, repeat until reach at middle. The next outermost elements (55 and 77) are swapped. The list is now 11, 77, 22, 55, and 33. There are no next outermost elements left, since the middle has been reached (22).

Animation captions:

1. Swap outermost elements.
2. Swap next outermost elements, repeat until reach at middle.

PARTICIPATION ACTIVITY 5.6.5: Reversing a list using swaps.

1) Using the above approach, how many swaps are needed to reverse this list:
999 888 777 666 555 444 333 222

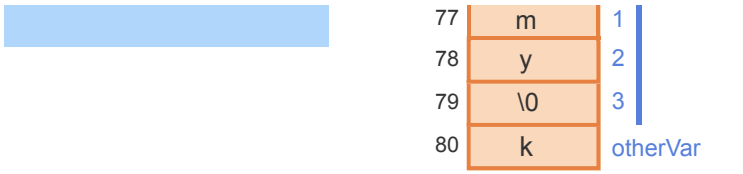
Check Show answer

5.7 Char arrays / C strings

A programmer can use an array to store a sequence of characters, known as a ***string***. Char arrays were the only kinds of strings in C++'s predecessor language C, and thus are sometimes called ***C strings*** to distinguish them from C++'s string type. An example is: `char movieTitle[20] = "Star Wars";`. Because a string can be shorter than the character array, a string in a char array must end with a special character known as a ***null character***, written as '\0'. Given a string literal like "Star Wars", the compiler automatically appends a null character.

PARTICIPATION ACTIVITY 5.7.1: A char array declaration and initialization with null-terminated string.

```
char name[4] = "Amy";
```



Animation content:

Begin static figure:
Begin C++ code:
char name[4] = "Amy";
End C++ code.
The elements of name are shown as their representations in memory.
Position 74 in memory is set to A. Position 75 is set to m. Position 76 is set to y. Position 77 is set to \0. Position 78 represents another variable, and is set to k.
"strlen(): 3" and "4th element for \0" are displayed.

Step 1: The character array must be large enough to include the null character. The compiler automatically inserts the null character to indicate the end of a string.
"strlen(): 3" and "4th element for \0" are displayed. For each character in name, a position in memory is set. Position 74 represents the 0th index of name, and is set to A. Position 75 represents the 1rst index of name, and is set to m. Position 76 represents the 2nd index of name, and is set to y. Position 77 represents the null terminating character of name, and is set to \0.

Animation captions:

1. The character array must be large enough to include the null character. The compiler automatically inserts the null character to indicate the end of a string.

A char array of size 20 can store strings of lengths 0 to 19. The longest string is 19, not 20, since the null character must be stored.

If a char array is initialized when declared, then the char array's size may be omitted, as in `char userName[] = "Hellen";` . The compiler determines the size from the string literal, in this case 6 + 1 (for the null character), or 7.

An array of characters ending with a null character is known as a ***null-terminated string***.

Output streams automatically handles null-terminated strings, printing each character until reaching the null character that ends the string.

Figure 5.7.1: Printing stops when reaching the null character at each string's end.

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

int main() {
   char cityName[20] = "Forest Lake"; // Compiler appends null
char

   // In each cout, printing stops when reaching null char
   cout << "City:" << endl;           // Compiler appends null
char to "City:"
   cout << cityName << endl;

   return 0;
}
```

```
City:
Forest
Lake
```


PARTICIPATION ACTIVITY 5.7.2: Char array strings.

Indicate whether the array declaration and initialization are appropriate.

1) `char firstName[10] = "Henry";`
   - True
   - False

2) `char lastName[10] = "Michelson";`
   - True
   - False

3) `char favoriteMuseum[10] = "Smithsonian";`
   - True
   - False

4) Printing catBreed will print 19 characters.
   `char catBreed[20] = "Persian";`
   - True
   - False

After a string is declared, a programmer may not later assign the string as in `movieTitle = "Indiana Jones";`. That statement tries to assign a value to the char array variable

itself, rather than copying each character from the string on the right into the array on the left. Functions exist to *copy* strings, such as strcpy(), discussed elsewhere.

A programmer can traverse a string using a loop that stops when reaching the null character.

A common error is to loop for the string's array size rather than stopping at the null character. Such looping visits unused array elements beyond the null character. *An even worse common error is to loop beyond the last valid element, which visits memory locations that are not part of the array.* These errors are illustrated below. Notice the strange characters that are output as the contents of other memory locations are printed out; the program may also crash.

Figure 5.7.2: Traversing a C string.

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

int main() {
   char userStr[20] = "1234567890123456789";
// Input string
   int i;

   // Prompt user for string input
   cout << "Enter string (<20 chars): ";
   cin >> userStr;

   // Print string
   cout << endl << userStr << endl;

   // Look for '@'
   for (i = 0; userStr[i] != '\0'; ++i) {
      if (userStr[i] == '@') {
         cout << "Found '@'." << endl;
      }
   }
   cout << endl;


   // The following is an ERROR.
   // May print chars it shouldn't.
   // Problem: doesn't stop at null char.
   cout << "\""; // Print opening "
   for (i = 0; i < 20; ++i) { // Print each
char
      cout << userStr[i];
   }
   cout << "\"" << endl;      // Print closing
"


   // The following is an even WORSE ERROR.
   // Accesses beyond valid index range.
   // Program may crash.
   cout << "\""; // Print opening "
   for (i = 0; i < 30; ++i) {
      cout << userStr[i];
   }
   cout << "\"" << endl; // Print closing "

   return 0;
}
```



```
Enter string (<20 chars):
test@gmail.com

test@gmail.com
Found '@'.

"test@gmail.com6789"
"test@gmail.com6789P!"
```


The above output is machine and compiler dependent. Also, some values aren't printable so don't appear in the output.

PARTICIPATION ACTIVITY 5.7.3: C string errors.

Given the following char array declaration, which of the following code snippets are bad?

```
char userText[10] = "Car";
```

1)
```
for (i = 0; userText != '\0';
++i) {
   // Print userText[i]
}
```

- OK
- Bad

2)
```
for (i = 0; userText[i] != '\0';
++i) {
   // Print userText[i]
}
```

- OK
- Bad

3)
```
for (i = 0; i < 10; ++i) {
   // Print userText[i]
}
```

- OK
- Bad

4)
```
for (i = 0; i < 4; ++i) {
   // Print userText[i]
}
```

- OK
- Bad

5)
```
userText = "Bus";
```

- OK
- Bad

Yet another common error with C strings is for the program user to enter a string larger than the character array. That may cause the input statement to write to memory locations outside the array's locations, which may corrupt other parts of program or data, and typically causes the program to crash.

zyDE 5.7.1: Reading in a string too large for a C string.

Run the program, which simply reads an input string and prints it one character at a time. Then, lengthen the input string beyond 10 characters, and run again. The program *might* work, if the extra memory locations being assigned don't matter. Try larger and larger strings, and

see if the program fails (be sure to scroll to the bottom of the output to look for erroneous output or an error message).

Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     char userStr[10]; //
6     int i;
7
8     // Prompt user for st
9     cout << "Enter string
10    cin >> userStr;
11
12    // Print 1 char at a
13    cout << endl;
14    for (i = 0; userStr[i
15       cout << userStr[i]
```

Hello

Run

C string usage is fraught with common errors. C++ introduced its own string type, as in `string myString;` and accessible after `#include <string>`, to reduce those errors and increase programmer convenience. C strings are still used in some legacy code and are thus good to learn. C++ provides common functions for handling C strings, which can be used by including the following: `#include <cstring>`.

The following program is for illustration, showing how a string is made up of individual character elements followed by a null character. Normally a programmer would not create a string that way.

Figure 5.7.3: A C string is an array of characters, ending with the null character.

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

int main() {
   char nameArr[5];

   nameArr[0] = 'A';
   nameArr[1] = 'l';
   nameArr[2] = 'a';
   nameArr[3] = 'n';
   nameArr[4] = '\0'; // Null character

   cout << nameArr << endl;

   nameArr[4] = '!';          // Oops, overwrote null char
   cout << nameArr << endl; // *Might* still work

   return 0;
}
```

```
Alan
Alan!
```


When printing a string stored within a character array, each character within the array will be printed until the null character is reached. If the null character is omitted, the program would print whatever values are found in memory after the array, until a null character happens to be encountered. Omitting the null character is a serious logical error.

It just so happens that the null character '\0' has an ASCII encoding of 0. Many compilers initialize memory to 0s. As such, omitting the '\0' in the above program would not always cause erroneous execution. Like a nail in the road, that bug in your code is just waiting to wreak havoc.

PARTICIPATION ACTIVITY 5.7.4: C string without null character.

Given:

```
char userText[10];
userText[0] = 'C';
userText[1] = 'a';
userText[2] = 'r';
userText[3] = '\0';
...
userText[3] = 's';
```


1) The first four characters in userText are now: Cars.

   - True
   - False

2) The compiler generates an error, because element 3 is the null character and can't be overwritten.

- True
- False

3) Printing userText should work fine because the new string is 4 characters, which is still much less than the array size of 10.

- True
- False

5.8 C-String library functions

C++ provides functions for working with C strings, presented in the ***cstring*** library. To use those functions, the programmer starts with: `#include <cstring>`.

Some C string functions for *modifying* strings are summarized below.

Table 5.8.1: Some C string modification functions.

Given:

```
char orgName[100] = "United Nations";
char userText[20] = "UNICEF";
char targetText[10];
```

strcpy()	`strcpy(destStr, sourceStr)`
	
	Copies sourceStr (up to and including null character) to destStr.
		`strcpy(targetText, userText); // Copies "UNICEF" + null char`
		
		`// to targetText`
		`strcpy(targetText, orgName); // Error: "United Nations"`
		`// has > 10 chars`
		`targetText = orgName; // Error: Strings can't be`
		
		`// copied this way`

strncpy()	`strncpy(destStr, sourceStr, numChars)` Copies up to numChars characters.	`strncpy(orgName, userText, 6); // orgName is "UNICEF Nations"`
strcat()	`strcat(destStr, sourceStr)` Copies sourceStr (up to and including null character) to *end* of destStr (starting at destStr's null character).	`strcat(orgName, userText); // orgName is "United NationsUNICEF"`

strncat()	`strncat(destStr, sourceStr, numChars)`
	
	Copies up to numChars characters to destStr's end, then appends null character.
		`strcpy(targetText, "abc"); // targetText is "abc"`
		`strncat(targetText, "123456789", 3); // targetText is "abc123"`

For strcpy(), *a common error is to copy a source string that is too large, causing an out-of-range access in the destination string. Another common error is to call strcpy with the source string first rather than the destination string, which copies in the wrong direction.*

Note that string assignment, as in targetText = orgName, does not copy the string and should not be used. The exception is during initialization, as in `char userText[20] = "UNICEF";`, for which the compiler copies the string literal's characters into the array.

PARTICIPATION ACTIVITY 5.8.1: String modification functions.

Given: char userStr[5];
Do not type quotes in your answers.
If the function call is incorrect, causes an out-of-range access, or the resulting string does not end with a null character, type Error.

1) What is userStr after: strcpy(userStr, "Bye");

Check Show answer

2) If userStr is initially "Hi", what is userStr after: strcpy(userStr, "Bye");

Check Show answer

3) What is userStr after: strcpy(userStr, "Goodbye");

Check Show answer

4) If userStr is initially "Hi!!", what is userStr after: strncpy(userStr, "Bye", 3);

Check Show answer

5) What is userStr after: strncpy(userStr, "Goodbye", 5);

Check Show answer

6) If userStr is initially "Hi", what is userStr after: strcat(userStr, '!');

Check Show answer

7) If userStr is initially "Hi", what is userStr after: strcat(userStr, "!");

Check Show answer

8) If userStr is initially "Hi", what is userStr after: strncat(userStr, "?!$#@%", 2);

Check Show answer

Several C string functions that get *information* about strings are summarized below.

Table 5.8.2: Some C string information functions.

Given:

```
char orgName[100] = "United Nations";
char userText[20] = "UNICEF";
char targetText[10];
```

strchr()	`strchr(sourceStr, searchChar)`
	
	Returns NULL if searchChar does not exist in sourceStr. (Else, returns address of first occurrence, discussed elsewhere).
	NULL is defined in the cstring library.
		`if (strchr(orgName, 'U') != NULL) { // 'U' exists in orgName?`
		`  ... // 'U' exists in "United Nations", branch taken`
		`}`
		`if (strchr(orgName, 'u') != NULL) { // 'u' exists in orgName?`
		`  ... // 'u' doesn't exist (case matters), branch not taken`
		`}`

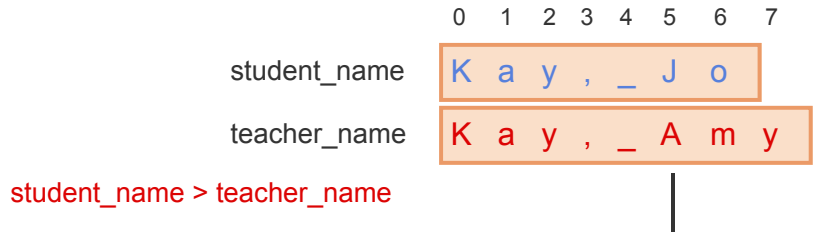
strlen()	`size_t strlen(sourceStr)`
	
	Returns number of characters in sourceStr up to, but not including, first null character. size_t is integer type.
		`x = strlen(orgName);  // Assigns 14 to x`
		`x = strlen(userText);  // Assigns 6 to x`
		`x = strlen(targetText); // Error: targetText may lack null char`

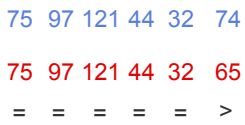
strcmp()	`int strcmp(str1, str2)`
	
	Returns 0 if str1 and str2 are equal, non-zero if they differ.
		`if (strcmp(orgName,"United Nations") == 0) {`
		`  ... // Equal, branch taken`
		`}`
		`if (strcmp(orgName, userText) != 0) {`
		`  ... // Not equal, branch taken`
		`}`

strcmp() is usually used to compare for equality, returning 0 if the strings are the same length and have identical characters. *A common error is to use == when comparing C strings, which does not work.* str1 == str2 compares the strings' addresses, not their contents. Because those addresses will usually be different, str1 == str2 will evaluate to 0. This is not a syntax error, but clearly a logic error. *Another common error is to forget to compare the result of strcmp with 0, as in* ***`if (strcmp(str1, str2)) {...}`***. The code is not a syntax error, but is a logic error because the if condition will be false (0) when the strings are equal. The correct condition would instead be `if (strcmp(str1, str2) == 0) {...}`. *Although strcmp returns 0, a good practice is to avoid using* ***`if (!strcmp(str1,str2)) {...}`*** *because that 0 does not represent "false" but rather is encoding a particular situation.*

strcmp(str1, str2) returns a negative number if str1 is less than str2, and a positive number if str1 is greater than str2. Evaluation first compares the character pair at element 0, then at element 1, etc., returning as soon as a pair differs.

PARTICIPATION ACTIVITY 5.8.2: String comparison.





Animation content:

Static Figure:
Two strings are shown, labeled "student_name" and "teacher_name".
"student_name" string is "Kay,_Jo" and has index values from 0-6.
"teacher_name" string is "Kay,_Amy" and has index values from 0 -7.
The ASCII values for each character for each string is shown.
The ASSII values for "student_name": 75, 97, 121, 44, 32, 74
The ASSII values for "teacher_name": 75, 97, 121, 44, 32, 65
Symbols are shown below the strings ASCII values that compare the ASCII character values: = = = = = >
The text "student_name > teacher_name" is shown to indicate that string "student_name" is greater than string "teacher_name".

Step 1: Each comparison uses ASCII values.
Strings "student_name" and "teacher_name" are shown. The first character at index 0 in each string is equal because both strings start with the character 'K' which is assigned the ASCII value 75.

Step 2: Values at indexes 0-4 are the same for both "student_name" and "teacher_name".
The second, third, fourth, and fifth characters at index's 1-4 in each string are equal because both strings contain characters, 'a', 'y', ', ', '_' which are assigned the ASCII values, 97, 121, 44, 32, consecutively.

Step 3: 'J' is greater than 'A', so student_name is greater than teacher_name.
The sixth character at index 5 in each string are not equal to one another. The character at index 5 in "student_name" is 'J' which is assigned to the ASCII value 74. The character at index 5 in "teacher_name" is 'A' which is assigned to the ASCII value 65. 'J' is greater than 'A', thus, the string "student_name" is greater than string "teacher_name"

Animation captions:

1. Each comparison uses ASCII values.
2. Values at indexes 0-4 are the same for both student_name and teacher_name.
3. 'J' is greater than 'A', so student_name is greater than teacher_name.

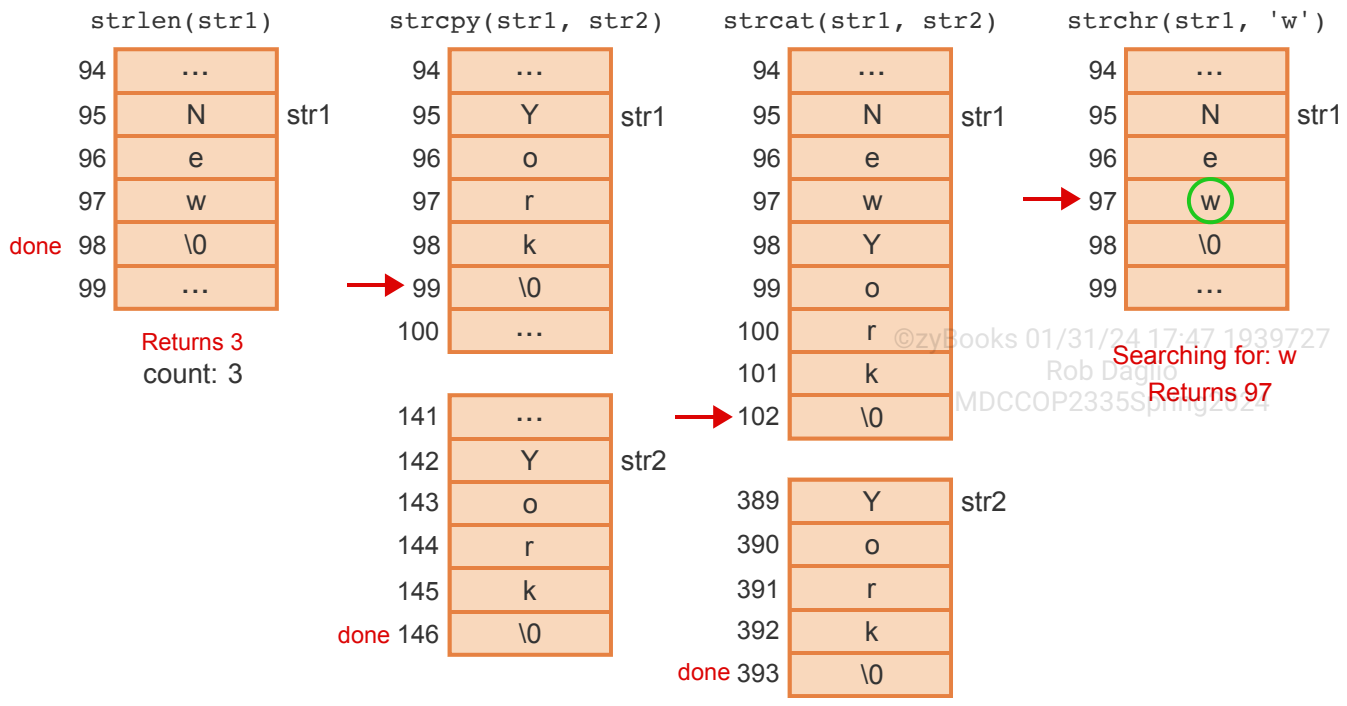
strlen is often used to iterate through each string character in a loop.

Figure 5.8.1: Iterating through a C string using strlen.

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

int main() {
   char userName[15] = "Alan Turing";
   int i;

   cout << "Before: " << userName <<
endl;

   for (i = 0; i < strlen(userName);
++i) {
      if (userName[i] == ' ') {
         userName[i] = '_';
      }
   }
   cout << "After:  " << userName <<
endl;

   return 0;
}
```

```
Before: Alan
Turing
After:
Alan_Turing
```


PARTICIPATION ACTIVITY 5.8.3: Some C string library functions.



Animation content:

Static Figure:
6 memory tables are shown.
The first address table consists of 6 memory addresses from 94 to 99. Memory address 94 and 99 contain "…". Memory address 95 contains the value 'N'. Memory address 96 contains the value 'e'. Memory address 97 contains the value 'w'. Memory address 98 contains the value '\0'.
The command line, "strlen(str1) " is shown above the first address table.

The second address table consists of 7 memory addresses from 94 to 100. Memory address 94 and 100 contain "…". Memory address 95 contains the value 'Y'. Memory address 96 contains the value 'o'. Memory address 97 contains the value 'r'. Memory address 98 contains the value 'k'. Memory address 99 contains the value '\0'.
The command line, "strcpy(str1, str2) " is shown above the second address table.

The third address table is positioned under the second table and consists of 6 memory addresses from 141 to 146. Memory address 141 contain "…". Memory address 142 contains the value 'Y'. Memory address 143 contains the value 'o'. Memory address 144 contains the value 'r'. Memory address 145 contains the value 'k'. Memory address 46 contains the value '\0'.

The fourth address table consists of 9 memory addresses from 94 to 102. Memory address 94 contain "…". Memory address 95 contains the value 'N'. Memory address 96 contains the value 'e'. Memory address 97 contains the value 'w'. Memory address 98 contains the value 'Y'. Memory address 99 contains the value 'o'. Memory address 100 contains the value 'r'. Memory address 101 contains the value 'k'. Memory address 102 contains the value '\0'.
The command line, "strcat(str1, str2) " is shown above the fourth address table.

The fifth address table is positioned under the fourth table and consists of 5 memory addresses from 389 to 393. Memory address 389 contains the value 'Y'. Memory address 390 contains the value 'o'. Memory address 391 contains the value 'r'. Memory address 392 contains the value 'k'. Memory address 393 contains the value '\0'.

The sixth address table consists of 6 memory addresses from 94 to 99. Memory address 94 and 99 contain "…". Memory address 95 contains the value 'N'. Memory address 96 contains the value 'e'. Memory address 97 contains the value 'w'. A green circle is shown around 'w'. Memory address 98 contains the value '\0'.
The command line, "strchr(str1, ' w')" is shown above the sixth address table.

Step 1: The strlen() function returns the number of characters in str1 up to, but not including, the first null character.
An address table is shown with 6 memory addresses from 94 to 99. The table contains characters from the string "str1". Memory address 94 and 99 contain "…". Memory address 95 contains the value

'N'. Memory address 96 contains the value 'e'. Memory address 97 contains the value 'w'. Memory address 98 contains the value '\0'. The strlen(str1) function executes, checking each character in the memory addresses, only counting 3 characters from memory addresses 95 to 97 and stopping at the null character at memory address 98. The execution flow returns the number 3.

Step 2: The strcpy() function copies str2 to str1, up to and including str2's null character.
Two address tables are shown, the first with 7 memory addresses from 94 to 100 and the second with 6 memory address from 141 to 146. The first table contains characters from the string "str1". Memory address 94, 99, and 100 contain "…". Memory address 95 contains the value 'N'. Memory address 96 contains the value 'e'. Memory address 97 contains the value 'w'. Memory address 98 contains the value '\0'.
The second table contains characters from the string "str2". Memory address 141 contain "…". Memory address 142 contains the value 'Y'. Memory address 143 contains the value 'o'. Memory address 144 contains the value 'r'. Memory address 145 contains the value 'k'. Memory address 46 contains the value '\0'.
The strcpy(str1, str2) function executes, replacing the characters from the first memory address table 95 to 99 with the characters from the second table from 142 to 146. Thus, the first memory address table now contains the characters from the string "str2".

Step 3: The strcat() function starts at str1's null character, then copies str2 to str1, up to and including str2's null character.
Two address tables are shown, the first with 9 memory addresses from 94 to 100 and the second with 5 memory address from 389 to 393. The first table contains characters from the string "str1". Memory address 94 and 99 through 102 contain "…". Memory address 95 contains the value 'N'. Memory address 96 contains the value 'e'. Memory address 97 contains the value 'w'. Memory address 98 contains the value '\0'.
The second table contains characters from the string "str2". Memory address 389 contains the value 'Y'. Memory address 390 contains the value 'o'. Memory address 391 contains the value 'r'. Memory address 392 contains the value 'k'. Memory address 393 contains the value '\0'.
The strcat(str1, str2) function executes, adding the characters from the second address table 389 to 393 to the characters from the second address table from 94 to 102. The null character in the first address table if found at memory address 98. This is the address where the characters from "str2" are added. Thus, the first memory address table now contains the characters from the string "str2" in addition to "str1".

Step 4: The strchr() function returns the address of the first occurrence of 'w' in str1. If 'w' does not exist in str1, then the strchr() function returns NULL.
A address table consists of 6 memory addresses from 94 to 99. Memory address 94 and 99 contain "…". Memory address 95 contains the value 'N'. Memory address 96 contains the value 'e'. Memory address 97 contains the value 'w'. A green circle is shown around 'w'. Memory address 98 contains the value '\0'.
The command line, "strchr(str1, 'w')" is shown above the sixth address table. The "strchr(str1, ' w')"

function executes, searching for the character 'w'in the memory addresses. The character 'w' is found at memory address 97. The execution flow returns the number 97.

Animation captions:

1. The strlen() function returns the number of characters in str1 up to, but not including, the first null character.
2. The strcpy() function copies str2 to str1, up to and including str2's null character.
3. The strcat() function starts at str1's null character, then copies str2 to str1, up to and including str2's null character.
4. The strchr() function returns the address of the first occurrence of 'w' in str1. If 'w' does not exist in str1, then the strchr() function returns NULL.

PARTICIPATION ACTIVITY 5.8.4: String information functions.

Given:

```
char str1[10] = "Earth";
char str2[20] = "Earthlings";
char str3[15] = "Mars";
```

Answer the following questions. If appropriate, type: Error

1) What does strlen(str3) return?

Check Show answer

2) Is the branch taken? (Yes/No/Error)

```
if  (strchr(str1, '@') !=
NULL) {
   // Print "Found @"
}
```

Check Show answer

3) Is the branch taken? (Yes/No/Error)

```
if  (strchr(str1, 'E') !=
NULL) {
   // Print "Found E"
}
```

Check Show answer

4) Is the branch taken? (Yes/No/Error)

```
if  (strchr(str2, "Earth") !=
NULL) {
   // Print "Found Earth"
}
```

Check Show answer

5) Is the branch taken? (Yes/No/Error)

```
if (strcmp(str1, str2) == 0)
{
   // Print "strings are
equal"
}
```

Check Show answer

6) Is the branch taken? (Yes/No/Error)

```
if (str1 == str3) {
   // Print "strings are
equal"
}
```


Check Show answer

7) Finish the code to take the branch if str1 and str3 are equal.

```
if (strcmp(str1, str3)
   ) {
   // Strings are equal
}
```

Check Show answer

Exploring further:

- More C string functions from cplusplus.com

5.9 Using a loop to modify or copy an array

Modifying array elements

A program may need to modify elements while iterating through an array. The program below uses a loop to convert any negative array element value to 0.

Figure 5.9.1: Modifying an array during iteration example: Converting negatives to 0 program.

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

int main() {
   const int NUM_ELEMENTS = 8;  // Number of elements
   int userVals[NUM_ELEMENTS];  // User values
   int i;                       // Loop index

   // Prompt user to input values
   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   // Convert negatives to 0
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      if (userVals[i] < 0) {
         userVals[i] = 0;
      }
   }

   // Print numbers
   cout << "New numbers: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << " ";
   }

   return 0;
}
```



```
Enter 8 integer
values...
Value: 5
Value: 67
Value: -5
Value: -4
Value: 5
Value: 6
Value: 6
Value: 4
New numbers: 5 67 0 0 5
6 6 4
```

PARTICIPATION ACTIVITY 5.9.1: Modifying an array in a loop.

What is the resulting array contents, assuming each question starts with an array of size 4 having contents -55, -1, 0, 9?

1)
```
for (i = 0; i < 4; ++i) {
   itemsList[i] = i;
}
```

- ○ -54, 0, 1, 10
- ○ 0, 1, 2, 3
- ○ 1, 2, 3, 4

2)
```
for (i = 0; i < 4; ++i) {
   if (itemsList[i] < 0) {
      itemsList[i] =
itemsList[i] * -1;
   }
}
```

- -55, -1, 0, -9
- 55, 1, 0, -9
- 55, 1, 0, 9

3)
```
for (i = 0; i < 4; ++i) {
   itemsList[i] =
itemsList[i+1];
}
```

- -1, 0, 9, 0
- 0, -55, -1, 0
- Out-of-range access

4)
```
for (i = 0; i < 3; ++i) {
   itemsList[i] =
itemsList[i+1];
}
```

- -1, 0, 9, 9
- Out-of-range access
- -1, 0, 9, 0

5)
```
for (i = 0; i < 3; ++i) {
   itemsList[i+1] =
itemsList[i];
}
```

- -55, -55, -55, -55
- -55, -55, -1, 0
- Out-of-range access

zyDE 5.9.1: Modifying an array during iteration example: Doubling element values.

Complete the following program to double each number in the array.

Load default template...

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

int main() {
   const int NUM_ELEMENT
   int userVals[NUM_ELEM
   int i;

   // Prompt user to inp
   cout << "Enter " << N
   for (i = 0; i < NUM_E
      cout << "Value: "
      cin >> userVals[i
   }
```

5 67 -5 -4 5 6 6 4

Run

Copying an array

Copying an array is a common task. Given a second array of the same size, a loop can copy each element one-by-one. Modifications to either array do not affect the other.

Figure 5.9.2: Array copying: Converting negatives to 0 program.

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

int main() {
   const int NUM_ELEMENTS = 8;    // Number of elements
   int userVals[NUM_ELEMENTS];    // User numbers
   int copiedVals[NUM_ELEMENTS]; // Copied/modified user
numbers
   int i;                         // Loop index

   // Prompt user for input values
   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   // Copy userVals to copiedVals array
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      copiedVals[i] =  userVals[i];
   }

   // Convert negatives to 0
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      if (copiedVals[i] < 0) {
         copiedVals[i] = 0;
      }
   }

   // Print numbers
   cout << endl << "Original and new values: " << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << " became " << copiedVals[i]
<< endl;
   }

   return 0;
}
```



```
Enter 8 integer
values...
Value: 12
Value: -5
Value: 34
Value: 75
Value: -14
Value: 33
Value: 12
Value: -104

Original and new
values:
12 became 12
-5 became 0
34 became 34
75 became 75
-14 became 0
33 became 33
12 became 12
-104 became 0
```

PARTICIPATION ACTIVITY 5.9.2: Array copying.

Given array firstList with size 4 and element values, 33, 44, 55, 66, and array secondList with size 4 and elements values 0, 0, 0, 0.

1) firstList = secondList copies 0s into each firstList element.

   - ○ True
   - ○ False

2) This loop copies firstList to secondList, so that secondList becomes 33, 44, 55, 66:

```
for (i = 0; i < 4; ++i) {
   secondList[i] = firstList[i];
}
```

- True
- False

3) After a for loop copies firstList to secondList, the assignment secondList[0] = 99 will modify both arrays.

- True
- False

4) Given thirdList with size 5 and elements 22, 21, 20, 19, 18, the following causes firstList's values to be 22, 21, 20, 19, 18:

```
for (i = 0; i < 5; ++i) {
   firstList[i] = thirdList[i];
}
```

- True
- False

CHALLENGE ACTIVITY 5.9.1: Decrement array elements.

Write a loop that subtracts 1 from each element in lowerScores. If the element was already 0 or negative, assign 0 to the element. Ex: lowerScores = {5, 0, 2, -3} becomes {4, 0, 1, 0}.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     const int SCORES_SIZE = 4;
6     int lowerScores[SCORES_SIZE];
7     int i;
8
9     for (i = 0; i < SCORES_SIZE; ++i) {
10       cin >> lowerScores[i];
11    }
12
```

```
13    /* Your solution goes here  */
14
15    for (i = 0; i < SCORES_SIZE; ++i) {
```

Run

CHALLENGE ACTIVITY 5.9.2: Copy and modify array elements.

Write a loop that sets newScores to oldScores rotated once left, with element 0 copied to the end. Ex: If oldScores = {10, 20, 30, 40}, then newScores = {20, 30, 40, 10}.

Note: These activities may test code with different test values. This activity will perform two tests, both with a 4-element array (int oldScores[4]). See "How to Use zyBooks".

Also note: If the submitted code tries to access an invalid array element, such as newScores[9] for a 4-element array, the test may generate strange results. Or the test may crash and report "Program end never reached", in which case the system doesn't print the test case that caused the reported message.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     const int SCORES_SIZE = 4;
6     int oldScores[SCORES_SIZE];
7     int newScores[SCORES_SIZE];
8     int i;
9
10    for (i = 0; i < SCORES_SIZE; ++i) {
11       cin >> oldScores[i];
12    }
13
14    /* Your solution goes here  */
15
```


Run

CHALLENGE ACTIVITY 5.9.3: Modify array elements using other elements.

Write a loop that sets each array element to the sum of itself and the next element, except for the last element which stays the same. Be careful not to index beyond the last element. Ex:

```
Initial scores:        10, 20, 30, 40
Scores after the loop: 30, 50, 70, 40
```

The first element is 30 or 10 + 20, the second element is 50 or 20 + 30, and the third element is 70 or 30 + 40. The last element remains the same.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     const int SCORES_SIZE = 4;
6     int bonusScores[SCORES_SIZE];
7     int i;
8
9     for (i = 0; i < SCORES_SIZE; ++i) {
10       cin >> bonusScores[i];
11    }
12
13    /* Your solution goes here  */
14
15    for (i = 0; i < SCORES_SIZE; ++i) {
```

Run

CHALLENGE ACTIVITY 5.9.4: Modify an array's elements.

Double any element's value that is less than controlValue. Ex: If controlValue = 10, then dataPoints = {2, 12, 9, 20} becomes {4, 12, 18, 20}.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     const int NUM_POINTS = 4;
6     int dataPoints[NUM_POINTS];
7     int controlValue;
8     int i;
9
10    cin >> controlValue;
11
12    for (i = 0; i < NUM_POINTS; ++i) {
13       cin >> dataPoints[i];
```

```
14    }
15
```

Run

5.10 Debugging example: Reversing an array

A common array modification is to reverse an array's elements. One way to accomplish this goal is to perform a series of swaps. For example, starting with an array of numbers 10 20 30 40 50 60 70 80, we could first swap the first item with the last item, yielding 80 20 30 40 50 60 70 10. We could next swap the second item with the second-to-last item, yielding 80 70 30 40 50 60 20 10. The next swap would yield 80 70 60 40 50 30 20 10, and the last would yield 80 70 60 50 40 30 20 10.

With this basic idea of how to reverse an array, we can attempt to write a program to carry out such reversal. Below we develop such a program but we make common mistakes along the way, to aid learning from examples of what not to do.

A first attempt to write a program that reverses an array appears below:

Figure 5.10.1: First program attempt to reverse array: Invalid access out of array bounds.

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

int main() {
   const int NUM_ELEMENTS = 8; // Number of elements
   int userVals[NUM_ELEMENTS]; // User numbers
   int i;                      // Loop index

   // Prompt user to input values
   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   // Reverse array's elements
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      userVals[i] = userVals[NUM_ELEMENTS - i]; //
Swap
   }

   // Print numbers
   cout << endl << "New values: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << " ";
   }

   return 0;
}
```



```
Enter 8 integer values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values: 0 80 70 60 50
60 70 80
```

Something went wrong: The program did not reverse the array, and the first element was set to 0. Let's try to find the code that caused the problem.

The first and third for loops are fairly standard, so let's initially focus attention on the middle for loop that does the reversing. The swap statement inside that loop is `userVals[i] = userVals[NUM_ELEMENTS - i]`. When i is 0, the statement will execute `userVals[0] = userVals[8];`. However, userVals has size 8 and thus valid indices are 0..7. userVals[8] does not exist. The program should actually swap elements 0 and 7, then 1 and 6, etc. Thus, let's change the right-side index to `NUM_ELEMENTS - 1 - i`. The revised program is shown below.

Figure 5.10.2: Next program attempt to reverse an array: Doesn't reverse properly; we forgot to swap.

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

int main() {
   const int NUM_ELEMENTS = 8; // Number of elements
   int userVals[NUM_ELEMENTS]; // User numbers
   int i;                      // Loop index

   // Prompt user to input values
   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   // Reverse array's elements
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      userVals[i] = userVals[NUM_ELEMENTS - 1 - i];
// Swap
   }

   // Print numbers
   cout << endl << "New values: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << " ";
   }

   return 0;
}
```



```
Enter 8 integer values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values: 80 70 60 50 50
60 70 80
```

The last four elements are still wrong. To determine what went wrong, we can manually (i.e., on paper) trace the loop's execution.

- i is 0: userVals[0] = userVals[7]. Array now: 80 20 30 40 50 60 70 80.
- i is 1: userVals[1] = userVals[6]. Array now: 80 70 30 40 50 60 70 80.
- i is 2: userVals[2] = userVals[5]. Array now: 80 70 60 40 50 60 70 80.
- i is 3: userVals[3] = userVals[4]. Array now: 80 70 60 50 50 60 70 80.
- i is 4: userVals[4] = userVals[3]. Array now: 80 70 60 50 50 60 70 80. *Uh-oh, where did 40 go?*

We failed to actually swap the array elements, instead the code just copies values in one direction. We need to add code to properly swap. We add a variable tempVal to temporarily hold `userVals[i]` so we don't lose that element's value.

Figure 5.10.3: Program with proper swap: However, the program's output shows the array doesn't change.

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

int main() {
   const int NUM_ELEMENTS = 8; // Number of elements
   int userVals[NUM_ELEMENTS]; // User numbers
   int i;                      // Loop index
   int tempVal;                // Temp variable for
swapping

   // Prompt user to input values
   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   // Reverse array's elements
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      tempVal = userVals[i];
// Temp for swap
      userVals[i] = userVals[NUM_ELEMENTS - 1 - i];
// First part of swap
      userVals[NUM_ELEMENTS - 1 - i] = tempVal;
// Second complete
   }

   // Print numbers
   cout << endl << "New values: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << " ";
   }

   return 0;
}
```



```
Enter 8 integer
values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values: 10 20 30 40
50 60 70 80
```

The new values are not reversed. Again, let's manually trace the loop iterations.

- i is 0: userVals[0] = userVals[7]. Array now: 80 20 30 40 50 60 70 10.
- i is 1: userVals[1] = userVals[6]. Array now: 80 70 30 40 50 60 20 10.
- i is 2: userVals[2] = userVals[5]. Array now: 80 70 60 40 50 30 20 10.
- i is 3: userVals[3] = userVals[4]. Array now: 80 70 60 50 40 30 20 10. *Looks reversed.*
- i is 4: userVals[4] = userVals[3]. Array now: 80 70 60 40 50 30 20 10. *Why are we still swapping?*

Tracing makes clear that the for loop should not iterate over the entire array. The reversal is completed halfway through the iterations. The solution is to set the loop expression to `i < (NUM_ELEMENTS / 2)`.

Figure 5.10.4: Program with correct loop bound: Running the program yields the correct output.

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

int main() {
   const int NUM_ELEMENTS = 8; // Number of elements
   int userVals[NUM_ELEMENTS]; // User numbers
   int i;                      // Loop index
   int tempVal;                // Temp variable for
swapping

   // Prompt user to input values
   cout << "Enter " << NUM_ELEMENTS << " integer
values..." << endl;
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << "Value: ";
      cin >> userVals[i];
   }

   // Reverse array's elements
   for (i = 0; i < (NUM_ELEMENTS / 2); ++i) {
      tempVal = userVals[i];
// Temp for swap
      userVals[i] = userVals[NUM_ELEMENTS - 1 - i];
// First part of swap
      userVals[NUM_ELEMENTS - 1 - i] = tempVal;
// Second complete
   }

   // Print numbers
   cout << endl << "New values: ";
   for (i = 0; i < NUM_ELEMENTS; ++i) {
      cout << userVals[i] << " ";
   }

   return 0;
}
```

```
Enter 8 integer
values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values: 80 70 60 50
40 30 20 10
```


We should ensure the program works if the number of elements is odd rather than even. Suppose the array has 5 elements (0-4) with values 10 20 30 40 50. `NUM_ELEMENTS / 2` would be 5 / 2 = 2, meaning the loop expression would be i < 2. The iteration when i is 0 would swap elements 0 and 4 (5-1-0), yielding 50 20 30 40 10. The iteration for i = 1 would swap elements 1 and 3, yielding 50 40 30 20 10. The loop would then not execute again because i is 2. So the results are correct for an odd number of elements, because the middle element will just not move.

The mistakes made above are each very common when dealing with loops and arrays, especially for beginning programmers. An incorrect (in this case out-of-range) index, an incorrect swap, and an incorrect loop expression. The lesson is that loops and arrays require attention to detail, greatly aided by

manually executing the loop to determine what is happening on each iteration. Ideally, a programmer will take more care when writing the original program, but the above mistakes are quite common.

PARTICIPATION ACTIVITY 5.10.1: Array reversal example.

Questions refer to the problematic example in this section.

1) The first problem was trying to access a non-existent element.
   - True
   - False

2) The second problem was failing to properly swap, using just this statement:

```
userVals[i] =
userVals[NUM_ELEMENTS - 1 - i];
// Swap
```

   - True
   - False

3) The third problem was that the loop did not iterate over all the elements, but rather stopped one short.
   - True
   - False

4) The programmer probably should have been more careful in creating the first version of the program.
   - True
   - False

5.11 C++ example: Annual salary tax rate calculation with vectors

zyDE 5.11.1: Various tax rates.

Vectors are useful to process tabular information. Income taxes are based on annual salary, usually with a tiered approach. Below is an example of a simple tax table:

Annual Salary	Tax Rate
0 to 20000	10%
Above 20000 to 50000	20%
Above 50000 to 100000	30%
Above 100000	40%

The below program uses a vector salaryBase to hold the cutoffs for each salary level and a parallel vector taxBase that has the corresponding tax rate.

1. Run the program and enter annual salaries of 40000 and 60000, then enter 0.
2. Modify the program to use two parallel vectors named annualSalaries and taxesToPay, each with 10 elements. Vectors annualSalaries holds up to 10 annual salaries entered; vector taxesToPay holds up to 10 corresponding amounts of taxes to pay for those annual salaries. Print the total annual salaries and taxes to pay after all input has been processed.
3. Run the program again with the same annual salary numbers as above.

Note: The calculation is inaccurate to how taxes are formally assessed and is a simplification for educational purposes only.

©zyBooks 01/31/24 17:47 1939727
Rob Daglio
MDCCOP2335Spring2024

Load default template...

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

int main() {
   const int MAX_ELEMENTS = 10;
   int annualSalary;
   double taxRate;
   int taxToPay;
   int numSalaries;
   bool keepLooking;
   unsigned int i;
   vector<int> salaryBase(5);
   vector<double> taxBase(5);
   // FIXME: Declare annualSalaries and taxesToPay ve
```


40000 60000 0

Run

zyDE 5.11.2: Various tax rates (solution).

A solution to the problem follows.

Load default template...

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

int main() {
   const int MAX_ELEMENTS = 10;
   int annualSalary;
   double taxRate;
   int taxToPay;
   int totalSalaries;
   int totalTaxes;
   int numSalaries;
   bool keepLooking;
   unsigned int i;
   int j;
```


40000 60000 0

Run

5.12 C++ example: Domain name validation with vectors

zyDE 5.12.1: Validate domain names with vectors.

Vectors are useful to process lists.

A ***top-level domain*** (TLD) name is the last part of an Internet domain name like .com in example.com. A ***core generic top-level domain*** (core gTLD) is a TLD that is either .com, .net, .org, or .info. A ***restricted top-level domain*** is a TLD that is either .biz, .name, or .pro. A ***second-level domain*** is a single name that precedes a TLD as in apple in apple.com.

The following program repeatedly prompts for a domain name, and indicates whether that domain name consists of a second-level domain followed by a core gTLD. Valid core gTLD's are stored in a vector. For this program, a valid domain name must contain only one period, such as apple.com, but not support.apple.com. The program ends when the user enters -1.

1. Run the program and enter domain names to validate.
2. Extend the program to also recognize restricted TLDs using a vector, and statements to validate against that vector. The program should also report whether the TLD is a core gTLD or a restricted gTLD. Run the program again.

Load default template...

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

int main() {
   // Define the list of valid core gTLDs
   const int NUM_ELEMENTS = 4;
   vector<string> validCoreGtld(NUM_ELEMENTS);
   // FIXME: Declare a vector named validRestrictedGt
   //        of the restricted domains, .biz, .name,
   string  inputName;
   string  searchName;
   string  theGtld;
   bool isValidDomainName;
   bool isCoreGtld;
```

```
apple.com
APPLE.com
apple.comm
```

Run

zyDE 5.12.2: Validate domain names with vectors (solution).

A solution to the problem follows.

Load default template...

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

int main() {
   // Define the list of valid core gTLDs
   const int NUM_ELEMENTS_CORE = 4;
   vector<string> validCoreGtld(NUM_ELEMENTS_CORE);
   const int NUM_ELEMENTS_RSTR = 3;
   vector<string> validRestrictedGtld(NUM_ELEMENTS_RS
   string  inputName;
   string  searchName;
   string  theGtld;
   bool isValidDomainName;
   bool isCoreGtld;
```



```
apple.com
APPLE.com
apple.comm
```

Run

5.13 Char library functions: cctype

C++ provides common functions for working with characters, presented in the cctype library. The first c indicates the library is a C language standard library, and ctype is short for "character type". To use those functions, the programmer adds the following at the top of a file: `#include <cctype>`

Commonly-used cctype functions are summarized below; a complete reference is found at the cctype reference page.

Character checking functions

The following functions check whether a character is of a given category, returning either false (0) or true (non-zero).

Table 5.13.1: Functions that check whether a character is of a given category.

The examples below assume the following string declaration.

```
char myString[30] = "Hey9! Go";
```


isalpha(c) -- Returns true if c is alphabetic: a-z or A-Z.	`isalpha('A');  // Returns true`
	`isalpha(myString[0]);  // Returns true because 'H' is alphabetic`
	`isalpha(myString[3]);  // Returns false because '9' is not alphabetic`

isdigit(c) -- Returns true if c is a numeric digit: 0-9.	`isdigit(myString[3]);  // Returns true because '9' is numeric`
	`isdigit(myString[4]);  // Returns false because ! is not numeric`

isalnum(c) -- Returns true if c is alphabetic or a numeric digit. Thus, returns true if either isalpha or isdigit would return true.	`isalnum('A');  // Returns true`
	`isalnum(myString[3]);  // Returns true because '9' is numeric`

isspace(c) -- Returns true if character c is a whitespace.	`isspace(myString[5]);  // Returns true because that character is a space ' '.`
	`isspace(myString[0]);  // Returns false because 'H' is not whitespace.`

islower(c) -- Returns true if character c is a lowercase letter a-z.	`islower(myString[0]);  // Returns false because 'H' is not lowercase.`
	`islower(myString[1]);  // Returns true because 'e' is lowercase.`
	`islower(myString[3]);  // Returns false because '9' is not a lowercase letter.`

isupper(c) -- Returns true if character c is an uppercase letter A-Z.	`isupper(myString[0]);  // Returns true because 'H' is uppercase.`
	`isupper(myString[1]);  // Returns false because 'e' is not uppercase.`
	`isupper(myString[3]);  // Returns false because '9' is not an uppercase letter.`

isblank(c) -- Returns true if character c is a blank character. Blank characters include spaces and tabs.	`isblank(myString[5]);  // Returns true because that character is a space ' '.`
	`isblank(myString[0]);  // Returns false because 'H' is not blank.`

isxdigit(c) -- Returns true if c is a hexadecimal digit: 0-9, a-f, A-F.	`isxdigit(myString[3]);  // Returns true because '9' is a hexadecimal digit.`
	`isxdigit(myString[1]);  // Returns true because 'e' is a hexadecimal digit.`
	`isxdigit(myString[6]);  // Returns false because 'G' is not a hexadecimal digit.`

ispunct(c) -- Returns true if c is a punctuation character. Punctuation characters include: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~	`ispunct(myString[4]);  // Returns true because '!' is a punctuation character.`
	`ispunct(myString[6]); // Returns false because 'G' is not a punctuation character.`

isprint(c) -- Returns true if c is a printable character. Printable characters include alphanumeric, punctuation, and space characters.	`isprint(myString[0]);  // Returns true because 'H' is a alphabetic.`
	`isprint(myString[4]);  // Returns true because '!' is punctuation.`
	`isprint(myString[5]);  // Returns true because that character is a space ' '.`
	`isprint('\0');  // Returns false because the null character is not printable`

iscntrl(c) -- Returns true if c is a control character. Control characters are all characters that are not printable.	`iscntrl(myString[0]);  // Returns false because 'H' is a not a control character`
	`iscntrl(myString[5]);  // Returns false because space is a not a control character`
	`iscntrl('\0');  // Returns true because the null character is a control character`

Character conversion functions

The following functions return a character representing a converted version of the input character.

Table 5.13.2: Functions that convert a character to upper or lower case.

The examples below assume the following string declaration.

```
char myString[30] = "Hey9! Go";
```

toupper(c) -- If c is a lowercase alphabetic character (a-z), returns the uppercase version (A-Z). If c is not a lowercase alphabetic character, just returns c.	`letter = toupper(myString[0]);  // Returns 'H' (no change)`
	`letter = toupper(myString[1]);  // Returns 'E' ('e' converted to 'E')`
	`letter = toupper(myString[3]);  // Returns '9' (no change)`
	`letter = toupper(myString[5]);  // Returns ' ' (no change)`

tolower(c) -- If c is an uppercase alphabetic character (A-Z), returns the lowercase version (a-z). If c is not an uppercase alphabetic character, just returns c.	`letter = tolower(myString[0]);  // Returns 'h' ('H' converted to 'h')`
	`letter = tolower(myString[1]);  // Returns 'e' (no change)`
	`letter = tolower(myString[3]);  // Returns '9' (no change)`
	`letter = tolower(myString[5]);  // Returns ' ' (no change)`

The following example illustrates some of the cctype functions.

Figure 5.13.1: Use of some functions in cctype.

```
Enter string (<30 chars):
ABC123$!def
Original: ABC123$!def
isalpha:  YYYNNNNNYYY
isdigit:  NNNYYYNNNNN
isupper:  YYYNNNNNNNN
After toupper: ABC123$!DEF
```


©zyBooks 01/31/24 17:47 1939727
Rob Daglio
MDCCOP2335Spring2024

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

int main() {
   const int MAX_LEN = 30;      // Max string
length
   char userStr[MAX_LEN];       // User defined
string
   int i;

   // Prompt user to enter string
   cout << "Enter string (<"
   << MAX_LEN << " chars): ";
   cin >> userStr;

   cout << "Original: " << userStr << endl;

   cout << "isalpha:  ";
   for (i = 0; userStr[i] != '\0'; ++i) {
      if (isalpha(userStr[i])) {
         cout << "Y";
      }
      else {
         cout << "N";
      }
   }
   cout << endl;

   cout << "isdigit:  ";
   for (i = 0; userStr[i] != '\0'; ++i) {
      if (isdigit(userStr[i])) {
         cout << "Y";
      }
      else {
         cout << "N";
      }
   }
   cout << endl;

   cout << "isupper:  ";
   for (i = 0; userStr[i] != '\0'; ++i) {
      if (isupper(userStr[i])) {
         cout << "Y";
      }
      else {
         cout << "N";
      }
   }
   cout << endl;

   for (i = 0; userStr[i] != '\0'; ++i) {
      userStr[i] = toupper(userStr[i]);
   }
   cout << "After toupper: " << userStr <<
endl;

   return 0;
}
```

To compare two strings without paying attention to case, one technique is to first convert (a copy of) each string to lowercase (using a loop, discussed elsewhere) and then comparing.

PARTICIPATION ACTIVITY 5.13.1: Character type functions.

Enter the value to which each function evaluates using 1 for true, 0 for false for boolean functions.
Assume str is "Hi 321!".

1) isalpha(str[0])

Check Show answer

2) isdigit(str[4])

Check Show answer

3) isalnum(str[2])

Check Show answer

4) isspace(str[2])

Check Show answer

5) islower(str[6])

Check Show answer

6) tolower(str[0])

Check Show answer

7) tolower(str[1])

Check Show answer

5.14 LAB: Output numbers in reverse

Write a program that reads a list of integers and outputs those integers in reverse. The input begins with an integer indicating the number of integers that follow. For coding simplicity, follow each output integer by a comma, including the last one.

Ex: If the input is:

```
5
2 4 6 8 10
```

the output is:

```
10,8,6,4,2,
```

Hint: First read the integers into a vector, then output the vector in reverse.

539740.3879454.qx3zqy7

LAB ACTIVITY | 5.14.1: LAB: Output numbers in reverse | 0 / 10

main.cpp Load default template...


```
1 #include <iostream>
2 #include <vector>   // Must include vector library to use vectors
3 using namespace std;
4
5 int main() {
6    vector<int> userInts; // A vector to hold the user's input integers
7
8    /* Type your code here. */
9
```

```
    return 0;
}
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

5.15 LAB: Middle item

Given a sorted list of integers, output the middle integer. A negative number indicates the end of the input (the negative number is not a part of the sorted list). Assume the number of integers is always odd.

Ex: If the input is:

```
2 3 4 8 11 -1
```

the output is:

```
Middle item: 4
```

The maximum number of list values for any test case should not exceed 9. If exceeded, output "Too many numbers".

Hint: First read the data into a vector. Then, based on the number of items, find the middle item.

LAB ACTIVITY 5.15.1: LAB: Middle item 0 / 10

main.cpp **Load default template...**

```
#include <iostream>
#include <vector>   // Must include vector library to use vectors
using namespace std;

int main() {

   /* Type your code here. */

   return 0;
}
```

Develop mode **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **main.cpp** (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

5.16 LAB: Output values below an amount

Write a program that first gets a list of integers from input. The input begins with an integer indicating the number of integers that follow. Then, get the last value from the input, which indicates a threshold. Output all integers less than or equal to that last threshold value.

Ex: If the input is:

```
5
50 60 140 200 75
100
```

the output is:

```
50,60,75,
```

The 5 indicates that there are five integers in the list, namely 50, 60, 140, 200, and 75. The 100 indicates that the program should output all integers less than or equal to 100, so the program outputs 50, 60, and 75.

For coding simplicity, follow every output value by a comma, including the last one.

Such functionality is common on sites like Amazon, where a user can filter results.

LAB ACTIVITY 5.16.1: LAB: Output values below an amount 0 / 10

main.cpp Load default template...

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6
7     /* Type your code here. */
8
9     return 0;
10 }
11
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working on this zyLab.
```

5.17 LAB: Adjust list by normalizing

When analyzing data sets, such as data for human heights or for human weights, a common step is to adjust the data. This adjustment can be done by normalizing to values between 0 and 1, or throwing away outliers.

For this program, adjust the values by dividing all values by the largest value. The input begins with an integer indicating the number of floating-point values that follow. Assume that the list will always contain positive floating-point values.

Output each floating-point value with two digits after the decimal point, which can be achieved by executing
`cout << fixed << setprecision(2);` once before all other cout statements.

Ex: If the input is:

```
5
30.0 50.0 10.0 100.0 65.0
```

the output is:

```
0.30 0.50 0.10 1.00 0.65
```


The 5 indicates that there are five floating-point values in the list, namely 30.0, 50.0, 10.0, 100.0, and 65.0. 100.0 is the largest value in the list, so each value is divided by 100.0.

For coding simplicity, follow every output value by a space, including the last one.

LAB ACTIVITY 5.17.1: LAB: Adjust list by normalizing 0 / 10

main.cpp Load default template...

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  using namespace std;
5
6  int main() {
7
8     /* Type your code here. */
9
10    return 0;
11 }
12
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.cpp** (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```


5.18 LAB: Word frequencies

Write a program that reads a list of words. Then, the program outputs those words and their frequencies. The input begins with an integer indicating the number of words that follow. Assume that the list will always contain fewer than 20 words.

Ex: If the input is:

```
5 hey hi Mark hi mark
```

the output is:

```
hey - 1
hi - 2
Mark - 1
hi - 2
mark - 1
```

Hint: Use two vectors, one vector for the strings and one vector for the frequencies.

LAB ACTIVITY | 5.18.1: LAB: Word frequencies | 0 / 10

main.cpp Load default template...

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 using namespace std;
5
6 int main() {
7
```

```
8     /* Type your code here. */
9
10    return 0;
11 }
12
```


Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.cpp** (Your program) → Output

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

5.19 LAB: Contains the character

Write a program that reads an integer, a list of words, and a character. The integer signifies how many words are in the list. The output of the program is every word in the list that contains the character at least once. For coding simplicity, follow each output word by a comma, even the last one. Add a newline to the end of the last output. Assume at least one word in the list will contain the given character.

Ex: If the input is:

```
4
hello zoo sleep drizzle
```

```
z
```

then the output is:

```
zoo,drizzle,
```

To achieve the above, first read the list into a vector. Keep in mind that the character 'a' is not equal to the character 'A'.

LAB ACTIVITY | 5.19.1: LAB: Contains the character | 0 / 10

main.cpp — Load default template...

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6
7     /* Type your code here. */
8
9     return 0;
10 }
11
```

Develop mode | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → **main.cpp** (Your program) → Output

Program output displayed here