

6.1 Recursion: Introduction

An **algorithm** is a sequence of steps for solving a problem. For example, an algorithm for making lemonade is:

Figure 6.1.1: Algorithms are like recipes.



Make lemonade:

- Add sugar to pitcher
- Add lemon juice
- Add water
- Stir

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Some problems can be solved using a recursive algorithm. A **recursive algorithm** is an algorithm that breaks the problem into smaller subproblems and applies the same algorithm to solve the smaller subproblems.

Figure 6.1.2: Mowing the lawn can be broken down into a recursive process.



- Mow the lawn
 - Mow the frontyard
 - Mow the left front
 - Mow the right front
 - Mow the backyard
 - Mow the left back
 - Mow the right back

The mowing algorithm consists of applying the mowing algorithm on smaller pieces of the yard and thus is a recursive algorithm.

At some point, a recursive algorithm must describe how to actually do something, known as the **base case**. The mowing algorithm could thus be written as:

- Mow the lawn
 - If lawn is less than 100 square meters
 - Push the lawnmower left-to-right in adjacent rows
 - Else
 - Mow one half of the lawn
 - Mow the other half of the lawn

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

PARTICIPATION
ACTIVITY

6.1.1: Recursion.



Which are recursive definitions/algorithms?



1) Helping N people:

If N is 1, help that person.
 Else, help the first N/2 people, then help the second N/2 people.

- True
- False

2) Driving to the store:

©zyBooks 06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

Go 1 mile.
 Turn left on Main Street.
 Go 1/2 mile.

- True
- False

3) Sorting envelopes by zipcode:



If N is 1, done.
 Else, find the middle zipcode. Put all zipcodes less than the middle zipcode on the left, all greater ones on the right.
 Then sort the left, then sort the right.

- True
- False

6.2 Recursive functions

A function may call other functions, including calling itself. A function that calls itself is a **recursive function**.

PARTICIPATION
ACTIVITY

6.2.1: A recursive function example.



Animation captions:

1. The first call to CountDown() function comes from main. Each call to CountDown() effectively creates a new "copy" of the executing function, as shown on the right.
2. Then, the CountDown() function calls itself. CountDown(1) similarly creates a new "copy" of the executing function.
3. CountDown() function calls itself once more.
4. That last instance does not call CountDown() again, but instead returns. As each instance returns, that copy is deleted.

06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

Each call to CountDown() effectively creates a new "copy" of the executing function, as shown on the right. Returning deletes that copy.

The example is for demonstrating recursion; counting down is otherwise better implemented with a loop.

Recursion may be direct, such as f() itself calling f(), or indirect, such as f() calling g() and g() calling f().

PARTICIPATION ACTIVITY

6.2.2: Thinking about recursion.



Refer to the above CountDown example for the following.

- 1) How many times is CountDown()
called if main() calls CountDown(5)?

Check
[Show answer](#)

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

- 2) How many times is CountDown()
called if main() calls CountDown(0)?

Check
[Show answer](#)

- 3) Is there a difference in how we
define the parameters of a recursive
versus non-recursive function?

Answer yes or no.

Check
[Show answer](#)
**CHALLENGE ACTIVITY**

6.2.1: Calling a recursive function.



Write a statement that calls the recursive function BackwardsAlphabet() with parameter startingLetter.

[Learn how our autograder works](#)

489394.3384924.qx3zqy7

```

1 #include <iostream>
2 using namespace std;
3
4 void BackwardsAlphabet(char currLetter){
5     if (currLetter == 'a') {
6         cout << currLetter << endl;
7     }
8     else{
9         cout << currLetter << " ";
10        BackwardsAlphabet(currLetter - 1);
11    }
12 }
13
14 int main(){
15     char startingLetter;

```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Run

View your last submission ▾

6.3 Recursive algorithm: Search

Recursive search (general)

Consider a guessing game program where a friend thinks of a number from 0 to 100 and you try to guess the number, with the friend telling you to guess higher or lower until you guess correctly. What algorithm would you use to minimize the number of guesses?

A first try might implement an algorithm that simply guesses in increments of 1:

- Is it 0? Higher
- Is it 1? Higher
- Is it 2? Higher

This algorithm requires too many guesses (50 on average). A second try might implement an algorithm that guesses by 10s and then by 1s:

- Is it 10? Higher
- Is it 20? Higher
- Is it 30? Lower
- Is it 21? Higher
- Is it 22? Higher
- Is it 23? Higher

This algorithm does better but still requires about 10 guesses on average: 5 to find the correct tens digit and 5 to guess the correct ones digit. An even better algorithm uses a binary search. A **binary search** algorithm begins at the midpoint of the range and halves the range after each guess. For example:

- Is it 50 (the middle of 0-100)? Lower
- Is it 25 (the middle of 0-50)? Higher
- Is it 38 (the middle of 26-50)? Lower
- Is it 32 (the middle of 26-38)?

After each guess, the binary search algorithm is applied again, but on a smaller range, i.e., the algorithm is recursive.

PARTICIPATION ACTIVITY

6.3.1: Binary search: A well-known recursive algorithm.



Animation content:

undefined

Animation captions:

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

1. A friend thinks of a number from 0 to 100 and you try to guess the number, with the friend telling you to guess higher or lower until you guess correctly.
2. Using a binary search algorithm, you begin at the midpoint of the lower range. ($\text{highVal} + \text{lowVal} / 2 = (100 + 0) / 2$, or 50).
3. The number is lower. The algorithm divides the range in half, then chooses the midpoint of that range.
4. After each guess, the binary search algorithm is applied, halving the range and guessing the midpoint or the corresponding range.

5. A recursive function is a natural match for the recursive binary search algorithm. A function GuessNumber(lowVal, highVal) has parameters that indicate the low and high sides of the guessing range.

Recursive search function

A recursive function is a natural match for the recursive binary search algorithm. A function GuessNumber(lowVal, highVal) has parameters that indicate the low and high sides of the guessing range. The function guesses at the midpoint of the range. If the user says lower, the function calls GuessNumber(lowVal, midVal). If the user says higher, the function calls GuessNumber(midVal + 1, highVal)

The recursive function has an if-else statement. The if branch ends the recursion, known as the **base case**. The else branch has recursive calls. Such an if-else pattern is common in recursive functions.

Figure 6.3.1: A recursive function carrying out a binary search algorithm.

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

void GuessNumber(int lowVal, int highVal) {
    int midVal; // Midpoint of low and high value
    char userAnswer; // User response

    midVal = (highVal + lowVal) / 2;

    // Prompt user for input
    cout << "Is it " << midVal << "? (l/h/y): ";
    cin >> userAnswer;

    if( (userAnswer != 'l') && (userAnswer != 'h') ) { // Base case: found
        number
        cout << "Thank you!" << endl;
    }
    else { // Recursive case:
        split into lower OR upper half
        if (userAnswer == 'l') { // Guess in lower half
            GuessNumber(lowVal, midVal); // Recursive call
        }
        else { // Guess in upper half
            GuessNumber(midVal + 1, highVal); // Recursive call
        }
    }
}

int main() {
    // Print game objective, user input commands
    cout << "Choose a number from 0 to 100." << endl;
    cout << "Answer with:" << endl;
    cout << "    l (your num is lower)" << endl;
    cout << "    h (your num is higher)" << endl;
    cout << "    any other key (guess is right)." << endl;

    // Call recursive function to guess number
    GuessNumber(0, 100);

    return 0;
}
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

```
Choose a number from 0 to 100.
Answer with:
    l (your num is lower)
    h (your num is higher)
    any other key (guess is right).
Is it 50? (l/h/y): l
Is it 25? (l/h/y): h
Is it 38? (l/h/y): l
Is it 32? (l/h/y): y
Thank you!
```

Calculating the middle value

Because `midVal` has already been checked, it need not be part of the new window, so `midVal + 1` rather than `midVal` is used for the window's new low side, or `midVal - 1` for the window's new high side. But the `midVal - 1` can have the drawback of a non-intuitive base case (i.e., `midVal < lowVal`, because if the current window is say 4..5, `midVal` is 4, so the new window would be 4..4-1, or 4..3). `rangeSize == 1` is likely more intuitive, and thus the algorithm uses `midVal` rather than `midVal - 1`. However, the algorithm uses `midVal + 1` when searching higher, due to integer rounding. In particular, for window 99..100, `midVal` is 99 ($(99 + 100) / 2 = 99.5$, rounded to 99 due to truncation of the fraction in integer division). So the next window would again be 99..100, and the algorithm would repeat

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

with this window forever. $\text{midVal} + 1$ prevents the problem, and doesn't miss any numbers because midVal was checked and thus need not be part of the window.

**PARTICIPATION
ACTIVITY**
6.3.2: Binary search tree tool.


The following program guesses the hidden number known by the user. Assume the hidden number is 63.

Books 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

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

void Find(int low, int high) {
    int mid; // Midpoint of low..high
    char answer;

    mid = (high + low) / 2;

    cout << "Is it " << mid << "? (l/h/y): ";
    cin >> answer;

    if((answer != 'l') &&
       (answer != 'h')) { // Base case:
        cout << "Thank you!" << endl; // Found number!
    }
    else { // Recursive case: Guess in
           // lower or upper half of range
        if (answer == 'l') { // Guess in lower half
            Find(low, mid); // Recursive call
        }
        else { // Guess in upper half
            Find(mid + 1, high); // Recursive call
        }
    }

    return;
}

int main() {
    cout << "Choose a number from 0 to 100." << endl;
    cout << "Answer with: " << endl;
    cout << "    l (your num is lower)" << endl;
    cout << "    h (your num is higher)" << endl;
    cout << "    any other key (guess is right)." << endl;

    Find(0, 100);

    return 0;
}
```

main()

```
int main() {
    cout << "Choose a number from 0 to 100." << endl;
    cout << "Answer with: " << endl;
    cout << "    l (your num is lower)" << endl;
    cout << "    h (your num is higher)" << endl;
    cout << "    any other key (guess is right)." << endl;

    Find(0, 100);

    return 0;
}
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Recursively searching a sorted list

Search is commonly performed to quickly find an item in a sorted list stored in an array or vector. Consider a list of attendees at a conference, whose names have been stored in alphabetical order in an array or vector. The following quickly determines whether a particular person is in attendance.

FindMatch() restricts its search to elements within the range lowVal to highVal. main() initially passes a range of the entire list: 0 to (list size - 1). FindMatch() compares to the middle element, returning that element's position if matching. If not matching, FindMatch() checks if the window's size is just one element, returning -1 in that case to indicate the item was not found. If neither of those two base cases are satisfied, then FindMatch() recursively searches either the lower or upper half of the range as appropriate.

Figure 6.3.2: Recursively searching a sorted list.

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

```

#include <iostream>
#include <string>
#include <vector>

using namespace std;

/* Finds index of string in vector of strings, else -1.
   Searches only with index range low to high
   Note: Upper/lower case characters matter
*/

int FindMatch(vector<string> stringsList, string itemMatch, int lowVal, int highVal) {
    int midVal;           // Midpoint of low and high values
    int itemPos;          // Position where item found, -1 if not found
    int rangeSize;        // Remaining range of values to search for match

    rangeSize = (highVal - lowVal) + 1;
    midVal = (highVal + lowVal) / 2;

    if (itemMatch == stringsList.at(midVal)) {    // Base case 1: item found
at midVal position
        itemPos = midVal;
    }
    else if (rangeSize == 1) {                      // Base case 2: match not
found
        itemPos = -1;
    }
    else {                                         // Recursive case: search
lower or upper half
        if (itemMatch < stringsList.at(midVal)) { // Search lower half,
recursive call
            itemPos = FindMatch(stringsList, itemMatch, lowVal, midVal);
        }
        else {                                     // Search upper half,
recursive call
            itemPos = FindMatch(stringsList, itemMatch, midVal + 1, highVal);
        }
    }
}

return itemPos;
}

int main() {
    vector<string> attendeesList(0); // List of attendees
    string attendeeName;           // Name of attendee to match
    int matchPos;                 // Matched position in attendee list

    // Omitting part of program that adds attendees
    // Instead, we insert some sample attendees in sorted order
    attendeesList.push_back("Adams, Mary");
    attendeesList.push_back("Carver, Michael");
    attendeesList.push_back("Domer, Hugo");
    attendeesList.push_back("Fredericks, Carlos");
    attendeesList.push_back("Li, Jie");

    // Prompt user to enter a name to find
    cout << "Enter person's name: Last, First: ";
    getline(cin, attendeeName); // Use getline to allow space in name

    // Call function to match name, output results
    matchPos = FindMatch(attendeesList, attendeeName, 0, attendeesList.size()
- 1);
    if (matchPos >= 0) {
        cout << "Found at position " << matchPos << "." << endl;
    }
    else {
        cout << "Not found. " << endl;
    }

    return 0;
}

```

@zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

```
Enter person's name: Last, First: Meeks, Stan
Not found.

...
Enter person's name: Last, First: Adams, Mary
Found at position 0.

...
Enter person's name: Last, First: Li, Jie
Found at position 4.
```

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

PARTICIPATION ACTIVITY

6.3.3: Recursive search algorithm.



Consider the above FindMatch() function for finding an item in a sorted list.

- 1) If a sorted list has elements 0 to 50 and the item being searched for is at element 6, how many times will FindMatch() be called?

Check
[Show answer](#)


- 2) If an alphabetically ascending list has elements 0 to 50, and the item at element 0 is "Bananas", how many calls to FindMatch() will be made during the failed search for "Apples"?

Check
[Show answer](#)

PARTICIPATION ACTIVITY

6.3.4: Recursive calls.



A list has 5 elements numbered 0 to 4, with these letter values: 0: A, 1: B, 2: D, 3: E, 4: F.

- 1) To search for item C, the first call is FindMatch(0, 4). What is the second call to FindMatch()?

- FindMatch(0, 0)
- FindMatch(0, 2)
- FindMatch(3, 4)



©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

- 2) In searching for item C, FindMatch(0, 2) is called. What happens next?

- Base case 1: item found at midVal.



- Base case 2: rangeSize == 1, so no match.
- Recursive call: FindMatch(2, 2)

CHALLENGE ACTIVITY

6.3.1: Enter the output of binary search.



489394.3384924.qx3zqy7

Start

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Type the program's output

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

void FindNumber(int number, int lowVal, int highVal) {
    int midVal;

    midVal = (highVal + lowVal) / 2;
    cout << number;
    cout << " ";
    cout << midVal;

    if (number == midVal) {
        cout << " q" << endl;
    }
    else {
        if (number < midVal) {
            cout << " r" << endl;
            FindNumber(number, lowVal, midVal);
        }
        else {
            cout << " s" << endl;
            FindNumber(number, midVal + 1, highVal);
        }
    }
}

int main() {
    int number;

    cin >> number;
    FindNumber(number, 0, 10);

    return 0;
}
```

Input

1

Output



1

2

Check**Next****CHALLENGE ACTIVITY**

6.3.2: Recursive algorithm: Search.



489394.3384924.qx3zqy7

Start

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Integer numData is read from input. Then, numData alphabetically sorted characters are read from input and appended to a vector. Complete the FindData() function, which outputs rangeSize, midIndex, and the element at index midIndex.

- Assign rangeSize with the total number of vector elements from lowerIndex to upperIndex (both inclusive).
- Assign midIndex with the result of dividing the sum of lowerIndex and upperIndex by 2.

Ex: If the input is:

```
5  
f h k m y
```

then the output is:

```
Number of elements in the range: 5  
Middle index: 2  
Element at middle index: k
```

```
1 #include <iostream>  
2 #include <vector>  
3 using namespace std;  
4  
5 void FindData(vector<char> lettersVector, int lowerIndex, int upperIndex) {  
6     int midIndex;  
7     int rangeSize;  
8  
9     /* Your code goes here */  
10  
11    cout << "Number of elements in the range: " << rangeSize << endl;  
12    cout << "Middle index: " << midIndex << endl;  
13    cout << "Element at middle index: " << lettersVector.at(midIndex) << endl;  
14 }  
15
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

1

2

Check

Next level

Exploring further:

- [Binary search](#) from GeeksforGeeks.org

6.4 Adding output statements for debugging

Recursive functions can be particularly challenging to debug. Adding output statements can be helpful. Furthermore, an additional trick is to indent the print statements to show the current depth of recursion. The following program adds a parameter indent to a `FindMatch()` function that searches a sorted list for an item. All of `FindMatch()`'s print statements start with `cout << indentAmt <<`. Indent is typically some number of spaces. `main()` sets indent to three spaces. Each recursive call adds three more spaces. Note how the output now clearly shows the recursion depth.

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Figure 6.4.1: Output statements can help debug recursive functions, especially if indented based on recursion depth.

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

```

#include <iostream>
#include <string>
#include <vector>

using namespace std;

/* Finds index of string in vector of strings, else -1.
   Searches only with index range low to high
   Note: Upper/lower case characters matter
*/

int FindMatch(vector<string> stringsList, string itemMatch,
              int lowVal, int highVal, string indentAmt) { // indentAmt used
for print debug

    int midVal;           // Midpoint of low and high values
    int itemPos;          // Position where item found, -1 if not found
    int rangeSize;        // Remaining range of values to search for match

    cout << indentAmt << "Find() range " << lowVal << " " << highVal << endl;

    rangeSize = (highVal - lowVal) + 1;
    midVal = (highVal + lowVal) / 2;

    if (itemMatch == stringsList.at(midVal)) {    // Base case 1: item found
at midVal position
        cout << indentAmt << "Found person." << endl;
        itemPos = midVal;
    }
    else if (rangeSize == 1) {                      // Base case 2: match not
found
        cout << indentAmt << "Person not found." << endl;
        itemPos = -1;
    }
    else {                                         // Recursive case: Search
lower or upper half
        if (itemMatch < stringsList.at(midVal)) { // Search lower half,
recursive call
            cout << indentAmt << "Searching lower half." << endl;
            itemPos = FindMatch(stringsList, itemMatch, lowVal, midVal,
indentAmt + " ");
        }
        else {                                     // Search upper half,
recursive call
            cout << indentAmt << "Searching upper half." << endl;
            itemPos = FindMatch(stringsList, itemMatch, midVal + 1, highVal,
indentAmt + " ");
        }
    }

    cout << indentAmt << "Returning pos = " << itemPos << "." << endl;
    return itemPos;
}

int main() {
    vector<string> attendeesList(0); // List of attendees
    string attendeeName;           // Name of attendee to match
    int matchPos;                 // Matched position in attendee list

    // Omitting part of program that adds attendees
    // Instead, we insert some sample attendees in sorted order
    attendeesList.push_back("Adams, Mary");
    attendeesList.push_back("Carver, Michael");
    attendeesList.push_back("Domer, Hugo");
    attendeesList.push_back("Fredericks, Carlos");
    attendeesList.push_back("Li, Jie");

    // Prompt user to enter a name to find
    cout << "Enter person's name: Last, First: ";
    getline(cin, attendeeName); // Use getline to allow space in name

    // Call function to match name, output results
    matchPos = FindMatch(attendeesList, attendeeName, 0);
}

```

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

```

Enter person's name: Last, First: Meeks, Stan
Find() range 0 4
Searching upper half.
Find() range 3 4
Searching upper half.
Find() range 4 4
Person not found.
Returning pos = -1.
Returning pos = -1.
Returning pos = -1.
Not found.

...
Enter person's name: Last, First: Adams, Mary
Find() range 0 4
Searching lower half.
Find() range 0 2
Searching lower half.
Find() range 0 1
Found person.
Returning pos = 0.
Returning pos = 0.
Returning pos = 0.
Found at position 0.

```

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

Some programmers like to leave the output statements in the code, commenting them out with "://" when not in use. The statements actually serve as a form of comment as well.

More advanced techniques for handling debug output exist too, such as **conditional compilation** (beyond this section's scope).

PARTICIPATION ACTIVITY

6.4.1: Recursive debug statements.



Refer to the above code using indented output statements.

- 1) The above debug approach requires an extra parameter be passed to indicate the amount of indentation.



- True
- False

- 2) Each recursive call should add a few spaces to the indent parameter.



- True
- False

- 3) The function should remove a few spaces from the indent parameter before returning.



- True
- False

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

zyDE 6.4.1: Output statements in a recursive function.

- Run the recursive program, and observe the output statements for debugging, and see if the person is correctly not found.

- Introduce an error by changing `itemPos = -1` to `itemPos = 0` in the range size base case.
- Run the program, notice how the indented print statements help isolate the error of person incorrectly being found.

[Load default template...](#)
[Run](#)

```

1 #include <iostream>
2 #include <string>
3 #include <vector>
4
5 using namespace std;
6
7 /* Finds index of string in vector
8    Searches only with index range l
9    Note: Upper/lower case character
10   */
11
12
13 int FindMatch(vector<string> string
14     .... int lowVal, int highV
15     ....

```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

6.5 Creating a recursive function

Creating a recursive function can be accomplished in two steps.

- **Write the base case** -- Every recursive function must have a case that returns a value without performing a recursive call. That case is called the **base case**. A programmer may write that part of the function first, and then test. There may be multiple base cases.
- **Write the recursive case** -- The programmer then adds the recursive case to the function.

The following illustrates a simple function that computes the factorial of N (i.e. $N!$). The base case is $N = 1$ or $1!$ which evaluates to 1. The base case is written as `if (N <= 1) { fact = 1; }`. The recursive case is used for $N > 1$, and written as `else { fact = N * NFact(N - 1); }`.

PARTICIPATION ACTIVITY

6.5.1: Writing a recursive function for factorial: First write the base case, then add the recursive case.



Animation captions:

1. The base case, which returns a value without performing a recursive call, is written and tested first. If N is less than or equal to 1, then the `NFact()` function returns 1.
2. Next the recursive case, which calls itself, is written and tested. If N is greater than 1, then the `NFact()` function returns $N * NFact(N - 1)$.

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

A common error is to not cover all possible base cases in a recursive function. Another common error is to write a recursive function that doesn't always reach a base case. Both errors may lead to infinite recursion, causing the program to fail.

Typically, programmers will use two functions for recursion. An "outer" function is intended to be called from other parts of the program, like the function `int CalcFactorial(int inVal)`. An "inner" function is intended only to be called from that outer function, for example a function `int CalcFactorialHelper(int inVal)`. The outer function may check for a valid input value, e.g., ensuring `inVal` is not negative, and then calling the inner function. Commonly, the inner function has

parameters that are mainly of use as part of the recursion, and need not be part of the outer function, thus keeping the outer function more intuitive.

**PARTICIPATION
ACTIVITY**
6.5.2: Creating recursion.


- 1) Recursive functions can be accomplished in one step, namely repeated calls to itself.
 - True
 - False

- 2) A recursive function with parameter N counts up from any negative number to 0. An appropriate base case would be $N == 0$.
 - True
 - False

- 3) A recursive function can have two base cases, such as $N == 0$ returning 0, and $N == 1$ returning 1.
 - True
 - False

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Before writing a recursive function, a programmer should determine:

1. Does the problem naturally have a recursive solution?
2. Is a recursive solution better than a non-recursive solution?

For example, computing $N!$ (N factorial) does have a natural recursive solution, but a recursive solution is not better than a non-recursive solution. The figure below illustrates how the factorial computation can be implemented as a loop. Conversely, binary search has a natural recursive solution, and that solution may be easier to understand than a non-recursive solution.

Figure 6.5.1: Non-recursive solution to compute $N!$

```
for (i = inputNum; i > 1; --i)
{
    facResult = facResult * i;
}
```

**PARTICIPATION
ACTIVITY**
6.5.3: When recursion is appropriate.

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

- 1) N factorial ($N!$) is commonly implemented as a recursive function due to being easier to understand and executing faster than a loop implementation.

- True

False

zyDE 6.5.1: Output statements in a recursive function.

Implement a recursive function to determine if a number is prime. Skeletal code is provided for the IsPrime function.

[Load default template...](#)
[Run](#)
©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

```

1 #include <iostream>
2 using namespace std;
3
4
5 // Returns false if value is not prime
6 bool IsPrime(int testVal, int divVal)
7 {
8     // Base case 1: 0 and 1 are not prime
9
10    // Base case 2: testVal only divisible by 1
11
12    // Recursive Case
13    // Check if testVal can be evenly divided by divVal
14    // Hint: use the % operator
15
16    ...
17 }
```

CHALLENGE ACTIVITY

6.5.1: Creating a recursive function.



489394.3384924.qx3zqy7

[Start](#)

Write MultiplySequence()'s base case to output " = " and result if inVal is less than or equal to 2. End with a colon. Ex: If the input is 6, then the output is:

6 * 4 * 2 = 48

```

1 #include <iostream>
2 using namespace std;
3
4 void MultiplySequence(int inVal, int result) {
5     cout << inVal;
6     result = result * inVal;
7
8     /* Your code goes here */
9
10    else {
11        cout << " * ";
12        MultiplySequence(inVal - 2, result);
13    }
14 }
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

1

2

[Check](#)[Next level](#)

6.6 Recursive math functions

Fibonacci sequence

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Recursive functions can solve certain math problems, such as computing the Fibonacci sequence. The **Fibonacci sequence** is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, etc.; starting with 0, 1, the pattern is to compute the next number by adding the previous two numbers.

Below is a program that outputs the Fibonacci sequence values step-by-step, for a user-entered number of steps. The base case is that the program has output the requested number of steps. The recursive case is that the program needs to compute the number in the Fibonacci sequence.

Figure 6.6.1: Fibonacci sequence step-by-step.

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

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

/* Output the Fibonacci sequence step-by-step.
Fibonacci sequence starts as:
0 1 1 2 3 5 8 13 21 ... in which the first
two numbers are 0 and 1 and each additional
number is the sum of the previous two numbers
*/

void ComputeFibonacci(int fibNum1, int fibNum2, int
runCnt) {

    cout << fibNum1 << " + " << fibNum2 << " = "
        << fibNum1 + fibNum2 << endl;

    if (runCnt <= 1) { // Base case: Ran for user
specified
                    // number of steps, do nothing
    }
    else {          // Recursive case: compute next
value
        ComputeFibonacci(fibNum2, fibNum1 + fibNum2,
runCnt - 1);
    }
}

int main() {
    int runFor;      // User specified number of
values computed

    // Output program description
    cout << "This program outputs the" << endl
        << "Fibonacci sequence step-by-step," << endl
        << "starting after the first 0 and 1." << endl <<
endl;

    // Prompt user for number of values to compute
    cout << "How many steps would you like? ";
    cin >> runFor;

    // Output first two Fibonacci values, call
recursive function
    cout << "0" << endl << "1" << endl;
    ComputeFibonacci(0, 1, runFor);

    return 0;
}
```

@zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

This program outputs the Fibonacci sequence step-by-step, starting after the first 0 and 1.

How many steps would you like? 10
0
1
0 + 1 = 1
1 + 1 = 2
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8
5 + 8 = 13
8 + 13 = 21
13 + 21 = 34
21 + 34 = 55
34 + 55 = 89

zyDE 6.6.1: Recursive Fibonacci.

Complete ComputeFibonacci() to return F_N , where F_0 is 0, F_1 is 1, F_2 is 1, F_3 is 2, F_4 is 3, and continuing: F_N is $F_{N-1} + F_{N-2}$. Hint: Base cases are $N == 0$ and $N == 1$.

@zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

```
1
2 #include <iostream>
3 using namespace std;
4
5 int ComputeFibonacci(int N) {
6
7     cout << "FIXME: Complete this function." << endl;
8     cout << "Currently just returns 0." << endl;
```

```
9      return 0;
10 }
11
12
13 int main() {
14     int N;          // F_N, starts at 0
15
16 }
```

Run

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Greatest common divisor (GCD)

Recursion can solve the greatest common divisor problem. The **greatest common divisor** (GCD) is the largest number that divides evenly into two numbers, e.g. $\text{GCD}(12, 8) = 4$. One GCD algorithm (described by Euclid around 300 BC) subtracts the smaller number from the larger number until both numbers are equal. Ex:

- $\text{GCD}(12, 8)$: Subtract 8 from 12, yielding 4.
- $\text{GCD}(4, 8)$: Subtract 4 from 8, yielding 4.
- $\text{GCD}(4, 4)$: Numbers are equal, return 4

The following recursively computes the GCD of two numbers. The base case is that the two numbers are equal, so that number is returned. The recursive case subtracts the smaller number from the larger number and then calls GCD with the new pair of numbers.

Figure 6.6.2: Calculate greatest common divisor of two numbers.

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

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

/* Determine the greatest common divisor
   of two numbers, e.g. GCD(8, 12) = 4
*/
int GCDcalculator(int inNum1, int inNum2) {
    int gcdVal; // Holds GCD results

    if(inNum1 == inNum2) { // Base case: Numbers are equal
        gcdVal = inNum1; // Return value
    }
    else { // Recursive case: subtract smaller from larger
        if (inNum1 > inNum2) { // Call function with new values
            gcdVal = GCDcalculator(inNum1 - inNum2,
inNum2);
        }
        else {
            gcdVal= GCDcalculator(inNum1, inNum2 -
inNum1);
        }
    }

    return gcdVal;
}

int main() {
    int gcdInput1; // First input to GCD calc
    int gcdInput2; // Second input to GCD calc
    int gcdOutput; // Result of GCD

    // Print program function
    cout << "Program outputs the greatest \n"
        << "common divisor of two numbers." << endl;

    // Prompt user for input
    cout << "Enter first number: ";
    cin >> gcdInput1;

    cout << "Enter second number: ";
    cin >> gcdInput2;

    // Check user values are > 1, call recursive GCD
    // function
    if ((gcdInput1 < 1) || (gcdInput2 < 1)) {
        cout << "Note: Neither value can be below 1." <<
endl;
    }
    else {
        gcdOutput = GCDcalculator(gcdInput1, gcdInput2);
        cout << "Greatest common divisor = " <<
gcdOutput << endl;
    }

    return 0;
}
```

@zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

Program outputs the greatest common divisor of two numbers.
 Enter first number: 12
 Enter second number: 8
 Greatest common divisor = 4

...

Program outputs the greatest common divisor of two numbers.
 Enter first number: 456
 Enter second number:
 784
 Greatest common divisor = 8

...

Program outputs the greatest common divisor of two numbers.
 Enter first number: 0
 Enter second number: 10
 Note: Neither value can be below 1.

@zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

PARTICIPATION ACTIVITY

6.6.1: Recursive GCD example.



- 1) How many calls are made to GCDcalculator() function for input values 12 and 8?



- 1
- 2
- 3

2) What is the base case for the GCD algorithm?

- When both inputs to the function are equal.
- When both inputs are greater than 1.
- When $\text{inNum1} > \text{inNum2}$.

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Exploring further:

- [Fibonacci number](#) from Wolfram.
- [Greatest Common Divisor](#) from Wolfram.

CHALLENGE ACTIVITY

6.6.1: Writing a recursive math function.



Write code to complete `RaiseToPower()`. Sample output if `userBase` is 4 and `userExponent` is 2 is shown below. Note: This example is for practicing recursion; a non-recursive function, or using the built-in function `pow()`, would be more common.

$4^2 = 16$

[Learn how our autograder works](#)

489394.3384924.qx3zqy7

```
1 #include <iostream>
2 using namespace std;
3
4 int RaiseToPower(int baseVal, int exponentVal){
5     int resultVal;
6
7     if (exponentVal == 0) {
8         resultVal = 1;
9     }
10    else {
11        resultVal = baseVal * /* Your solution goes here */;
12    }
13
14    return resultVal;
15 }
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Run

View your last submission ▾

6.7 Recursive exploration of all possibilities

Recursion is a powerful technique for exploring all possibilities, such as all possible reorderings of a word's letters, all possible subsets of items, all possible paths between cities, etc. This section provides several examples.

Word scramble

Consider printing all possible combinations (or "scramblings") of a word's letters. The letters of abc can be scrambled in 6 ways: abc, acb, bac, bca, cab, cba. Those possibilities can be listed by making three choices: Choose the first letter (a, b, or c), then choose the second letter, then choose the third letter. The choices can be depicted as a tree. Each level represents a choice. Each node in the tree shows the unchosen letters on the left, and the chosen letters on the right.

PARTICIPATION
ACTIVITY

6.7.1: Exploring all possibilities viewed as a tree of choices.



Animation content:

Static figure: A tree of choices with 4 levels and 16 nodes. Each node contains the remaining letters and chosen letters separated with a /. The root node contains abc/ and has 3 children. The second level containing 3 nodes has the label "Choose first letter". From left to right, the nodes contain bc/a, ac/b, and ab/c, respectively. Each node on the second level has 2 children. The third level containing 6 nodes has the label "Choose second letter". From left to right, the nodes contain c/ab, b/ac, c/ba, a/bc, b/ca, and a/cb, respectively. Each node on the third level has 1 child. The fourth level containing 6 nodes has the label "Choose third letter". From left to right, the nodes contain /abc, /acb, /bac, /bca, /cab, and /cba, respectively.

Animation captions:

1. Consider printing all possible combinations of a word's letters. Those possibilities can be listed by choosing the first letter, then the second letter, then the third letter.
2. The choices can be depicted as a tree. Each level represents a choice.
3. A recursive exploration function is a natural match to print all possible combinations of a string's letters. Each call to the function chooses from the set of unchosen letters, continuing until no unchosen letters remain.

The tree guides creation of a recursive exploration function to print all possible combinations of a string's letters. The function takes two parameters: unchosen letters, and already chosen letters. The base case is no unchosen letters, causing printing of the chosen letters. The recursive case calls the function once for each letter in the unchosen letters. The above animation depicts how the recursive algorithm traverses the tree. The tree's leaves (the bottom nodes) are the base cases.

The following program prints all possible ordering of the letters of a user-entered word.

Figure 6.7.1: Scramble a word's letters in every possible way.

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

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

/* Output every possible combination of a word.
   Each recursive call moves a letter from
   remainLetters to scramLetters.
*/
void ScrambleLetters(string remainLetters, // Remaining
                      letters
                      string scramLetters) { // Scrambled
    letters
    string tmpString; // Temp word combination
    unsigned int i; // Loop index

    if (remainLetters.size() == 0) { // Base case: All
        letters used
        cout << scramLetters << endl;
    }
    else { // Recursive case:
        move a letter from
        // remaining to
        scrambled letters
        for (i = 0; i < remainLetters.size(); ++i) {
            // Move letter to scrambled letters
            tmpString = remainLetters.substr(i, 1);
            remainLetters.erase(i, 1);
            scramLetters = scramLetters + tmpString;

            ScrambleLetters(remainLetters, scramLetters);

            // Put letter back in remaining letters
            remainLetters.insert(i, tmpString);
            scramLetters.erase(scramLetters.size() - 1, 1);
        }
    }
}

int main() {
    string wordScramble; // User defined word to scramble

    // Prompt user for input
    cout << "Enter a word to be scrambled: ";
    cin >> wordScramble;

    // Call recursive function
    ScrambleLetters(wordScramble, "");

    return 0;
}
```

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

Enter a word to be
 scrambled: cat
 cat
 cta
 act
 atc
 tca
 tac

PARTICIPATION ACTIVITY

6.7.2: Letter scramble.



- 1) What is the output of
 ScrambleLetters("xy", "")? Determine
 your answer by manually tracing the
 code, not by running the program.

- xy xy
- xx yy xy yx
- xy yx

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

Shopping spree

Recursion can find all possible subsets of a set of items. Consider a shopping spree in which a person can select any 3-item subset from a larger set of items. The following program prints all possible 3-item subsets of a given larger set. The program also prints the total price of each subset.

ShoppingBagCombinations() has a parameter for the current bag contents, and a parameter for the remaining items from which to choose. The base case is that the current bag already has 3 items, which prints the items. The recursive case moves one of the remaining items to the bag, recursively calling the function, then moving the item back from the bag to the remaining items.

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Figure 6.7.2: Shopping spree in which a user can fit 3 items in a shopping bag.

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

```
Milk   Belt
Toys  = $45
Milk   Belt
Cups   = $38
Milk   Toys
Belt   = $45
Milk   Toys
Cups   = $33
Milk   Cups
Belt   = $38
Milk   Cups
Toys   = $33
Belt   Milk
Toys   = $45
Belt   Milk
Cups   = $38
Belt   Toys
Milk   = $45
Belt   Toys
Cups   = $55
Belt   Cups
Milk   = $38
Belt   Cups
Toys   = $55
Toys   Milk
Belt   = $45
Toys   Milk
Cups   = $33
Toys   Belt
Milk   = $45
Toys   Belt
Cups   = $55
Toys   Cups
Milk   = $33
Toys   Cups
Belt   = $55
Cups   Milk
Belt   = $38
Cups   Milk
Toys   = $33
Cups   Belt
Milk   = $38
Cups   Belt
Toys   = $55
Cups   Toys
Milk   = $33
Cups   Toys
Belt   = $55
```

zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

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

class Item {
public:
    string itemName; // Name of item
    int priceDollars; // Price of item
};

const unsigned int MAX_ITEMS_IN_SHOPPING_BAG = 3; // Max num items

/* Output every combination of items that fit
   in a shopping bag. Each recursive call moves
   one item into the shopping bag.
*/
void ShoppingBagCombinations(vector<Item> currBag,           // Bag contents
                               vector<Item> remainingItems) { // Available items
    int bagValue;           // Cost of items in shopping bag
    Item tmpGroceryItem; // Grocery item to add to bag
    unsigned int i;         // Loop index

    if (currBag.size() == MAX_ITEMS_IN_SHOPPING_BAG) { // Base case: Shopping bag full
        bagValue = 0;
        for (i = 0; i < currBag.size(); ++i) {
            bagValue += currBag.at(i).priceDollars;
            cout << currBag.at(i).itemName << " ";
        }
        cout << "= $" << bagValue << endl;
    } else { // Recursive case: move one
        for (i = 0; i < remainingItems.size(); ++i) { // item to bag
            // Move item into bag
            tmpGroceryItem = remainingItems.at(i);
            remainingItems.erase(remainingItems.begin() + i);
            currBag.push_back(tmpGroceryItem);

            ShoppingBagCombinations(currBag, remainingItems);

            // Take item out of bag
            remainingItems.insert(remainingItems.begin() +
i, tmpGroceryItem);
            currBag.pop_back();
        }
    }
}

int main() {
    vector<Item> possibleItems(0); // Possible shopping items
    vector<Item> shoppingBag(0); // Current shopping bag
    Item tmpGroceryItem; // Temp item

    // Populate grocery with different items
    tmpGroceryItem.itemName = "Milk";
    tmpGroceryItem.priceDollars = 2;
    possibleItems.push_back(tmpGroceryItem);

    tmpGroceryItem.itemName = "Belt";
    tmpGroceryItem.priceDollars = 24;
    possibleItems.push_back(tmpGroceryItem);

    tmpGroceryItem.itemName = "Toys";
    tmpGroceryItem.priceDollars = 19;
    possibleItems.push_back(tmpGroceryItem);

    tmpGroceryItem.itemName = "Gum";
}
```

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462
 Taylor Larrechea
 COLORADOCSPB2270Summer2023

**PARTICIPATION
ACTIVITY**

6.7.3: All letter combinations.



- 1) When main() calls ShoppingBagCombinations(), how many items are in the remainingItems list?

- None
- 3
- 4

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

- 2) When main() calls ShoppingBagCombinations(), how many items are in currBag list?

- None
- 1
- 4

- 3) After main() calls ShoppingBagCombinations(), what happens first?

- The base case prints Milk, Belt, Toys.
- The function bags one item, makes recursive call.
- The function bags 3 items, makes recursive call.

- 4) Just before ShoppingBagCombinations() returns back to main(), how many items are in the remainingItems list?

- None
- 4

- 5) How many recursive calls occur before the first combination is printed?

- None
- 1
- 3

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

- 6) What happens if main() only put 2, rather than 4, items in the possibleItems list?

- Base case never executes; nothing printed.
- Infinite recursion occurs.



Traveling salesman

Recursion is useful for finding all possible paths. Suppose a salesman must travel to 3 cities: Boston, Chicago, and Los Angeles. The salesman wants to know all possible paths among those three cities, starting from any city. A recursive exploration of all travel paths can be used. The base case is that the salesman has traveled to all cities. The recursive case is to travel to a new city, explore possibilities, then return to the previous city.

Figure 6.7.3: Find distance of traveling to 3 cities.

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Boston	Chicago	Los
Angeles	= 2971	
Boston	Los Angeles	
Chicago	= 4971	
Chicago	Boston	Los
Angeles	= 3920	
Chicago	Los Angeles	
Boston	= 4971	
Los Angeles	Boston	
Chicago	= 3920	
Los Angeles	Chicago	
Boston	= 2971	

@zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

@zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

```

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

const unsigned int NUM_CITIES = 3;           // Number of
cities
int cityDistances[NUM_CITIES][NUM_CITIES]; // Distance
between cities
string cityNames[NUM_CITIES];               // City
names

/* Output every possible travel path.
   Each recursive call moves to a new city.
*/
void TravelPaths(vector<int> currPath, vector<int>
needToVisit) {
    int totalDist;      // Total distance given current
path
    int tmpCity;        // Next city distance
    unsigned int i;     // Loop index

    if (currPath.size() == NUM_CITIES) { // Base case:
Visited all cities
        totalDist = 0;                  // return
total path distance
        for (i = 0; i < currPath.size(); ++i) {
            cout << cityNames[currPath.at(i)] << " ";
            if (i > 0) {
                totalDist += cityDistances[currPath.at(i - 1)][currPath.at(i)];
            }
        }

        cout << "= " << totalDist << endl;
    }
    else {                      // Recursive
case: pick next city
        for (i = 0; i < needToVisit.size(); ++i) {
            // Add city to travel path
            tmpCity = needToVisit.at(i);
            needToVisit.erase(needToVisit.begin() + i);
            currPath.push_back(tmpCity);

            TravelPaths(currPath, needToVisit);

            // Remove city from travel path
            needToVisit.insert(needToVisit.begin() + i,
tmpCity);
            currPath.pop_back();
        }
    }
}

int main() {
    vector<int> needToVisit(0); // Cities left to visit
    vector<int> currPath(0);   // Current path traveled

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

    cityNames[0] = "Boston";
    cityNames[1] = "Chicago";
    cityNames[2] = "Los Angeles";
}

```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

PARTICIPATION ACTIVITY

6.7.4: Recursive exploration.



- 1) You wish to generate all possible 3-letter subsets from the letters in an N-letter word ($N > 3$). Which of the above recursive functions is the closest?

- ShoppingBagCombinations
- ScrambleLetters
- main()

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

CHALLENGE ACTIVITY

6.7.1: Enter the output of recursive exploration.



489394.3384924.qx3zqy7

Start

Type the program's output

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

void ScrambleNums(vector<int> remainNums, vector<int> scramNums) {
    vector<int> tmpRemainNums;
    int tmpRemovedNum;
    int i;

    if (remainNums.size() == 0) {
        cout << scramNums.at(0);
        cout << scramNums.at(1);
        cout << scramNums.at(2) << endl;
    }
    else {
        for (i = 0; i < remainNums.size(); ++i) {
            tmpRemainNums = remainNums; // Make a copy.
            tmpRemovedNum = tmpRemainNums.at(i);
            tmpRemainNums.erase(tmpRemainNums.begin() + i); // Remove element at i
            scramNums.push_back(tmpRemovedNum);
            ScrambleNums(tmpRemainNums, scramNums);
            scramNums.erase(scramNums.end() - 1); // Remove last element
        }
    }
}

int main() {
    vector<int> numsToScramble;
    vector<int> resultNums;

    numsToScramble.push_back(2);
    numsToScramble.push_back(9);
    numsToScramble.push_back(0);

    ScrambleNums(numsToScramble, resultNums);

    return 0;
}
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

1

2

Check**Next**

**CHALLENGE
ACTIVITY**

6.7.2: Recursive exploration of all possibilities.



489394.3384924.qx3zqy7

Start

Integer vectSize is read from input, then vectSize strings are read and stored into vector passengersTo write the base case to output each element in vector pickedPassengers if the size of vector remainPas space after each element. End with a newline.

@zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Ex: If the input is:

```
3
Kim Rob Zoe
```

then the output is:

```
All unique queues:
Kim Rob Zoe
Kim Zoe Rob
Rob Kim Zoe
Rob Zoe Kim
Zoe Kim Rob
Zoe Rob Kim
```

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 void OrderPassengers(vector<string> remainPassengers, vector<string> pickedPass
6     unsigned int i;
7     string pick;
8
9     /* Your code goes here */
10
11    else {
12        for (i = 0; i < remainPassengers.size(); ++i) {
13            pick = remainPassengers.at(i);
14            remainPassengers.erase(remainPassengers.begin() + i);
15            pickedPassengers.push_back(pick);
```

1

2

Check**Next level**

@zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Exploring further:

- [Recursive Algorithms](#) from khanacademy.org

6.8 Stack overflow

Recursion enables an elegant solution to some problems. But, for large problems, deep recursion can cause memory problems. Part of a program's memory is reserved to support function calls. Each function call places a new **stack frame** on the stack, for local parameters, local variables, and more function items. Upon return, the frame is deleted.

Deep recursion could fill the stack region and cause a **stack overflow**, meaning a stack frame extends beyond the memory region allocated for stack. Stack overflow usually causes the program to crash and report an error like: segmentation fault, access violation, or bad access.

PARTICIPATION ACTIVITY
6.8.1: Recursion causing stack overflow.
 ©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

Animation captions:

1. Deep recursion may cause stack overflow, causing a program to crash.

The animation showed a tiny stack region for easy illustration of stack overflow.

The size of parameters and local variables results in a larger stack frame. Large vectors, arrays, or strings declared as local variables, or passed by copy, can lead to faster stack overflow.

A programmer can estimate recursion depth and stack size to determine whether stack overflow might occur. Sometimes a non-recursive algorithm must be developed to avoid stack overflow.

PARTICIPATION ACTIVITY
6.8.2: Stack overflow.


- 1) A memory's stack region can store at most one stack frame.

- True
- False

- 2) The size of the stack is unlimited.

- True
- False

- 3) A stack overflow occurs when the stack frame for a function call extends past the end of the stack's memory.

- True
- False

- 4) The following recursive function will result in a stack overflow.

```
int RecAdder(int inValue) {
    return RecAdder(inValue + 1);
}
```

 ©zyBooks 06/06/23 23:52 1692462

Taylor Larrechea

COLORADOCSPB2270Summer2023

- True
- False

6.9 C++ example: Recursively output permutations

zyDE 6.9.1: Recursively output permutations.

The below program prints all permutations of an input string of letters, one permutation per line. Ex: The six permutations of "cab" are:

```
cab  
cba  
acb  
abc  
bca  
bac
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Below, the PermuteString function works recursively by starting with the first character and permuting the remainder of the string. The function then moves to the second character and permutes the string consisting of the first character and the third through the end of the string, and so on.

1. Run the program and input the string "cab" (without quotes) to see that the above output is produced.
2. Modify the program to print the permutations in the opposite order, and also to output the permutation count on each line.
3. Run the program again and input the string cab. Check that the output is reversed.
4. Run the program again with an input string of abcdef. Why did the program take longer to produce the results?

```
Load default template
```

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 // FIXME: Use a static variable to count permutations. Why should this be?
6
7 void PermuteString(string head, string tail) {
8     char current;
9     string newPermute;
10    int len;
11    int i;
12
13    current = '?';
14
15    len = tail.size();
16    for (i = 0; i < len; i++) {
17        newPermute = head + tail[i];
18        tail = tail.substr(0, i) + tail.substr(i+1);
19        PermuteString(newPermute, tail);
20    }
21}
```

```
cab
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Run

zyDE 6.9.2: Recursively output permutations (solution).

Below is the solution to the above problem.

[Load default template](#)

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 static int permutationCount = 0;
6
7 void PermuteString(string head, string tail) {
8     char current;
9     string newPermute;
10    int len;
11    int i;
12
13    current = '?';
14
15    len = tail.size();
16    if (len == 0) {
17        cout << head << endl;
18        permutationCount++;
19    } else {
20        for (i = 0; i < len; i++) {
21            current = tail[i];
22            tail.erase(i, 1);
23            newPermute = head + current + tail;
24            PermuteString(newPermute, tail);
25            tail.insert(i, 1, current);
26        }
27    }
28}
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

cab
abcdef

[Run](#)

6.10 Recursive definitions



This section has been set as optional by your instructor.

Recursive algorithms

An **algorithm** is a sequence of steps, including at least 1 terminating step, for solving a problem. A **recursive algorithm** is an algorithm that breaks the problem into smaller subproblems and applies the algorithm itself to solve the smaller subproblems.

Because a problem cannot be endlessly divided into smaller subproblems, a recursive algorithm must have a **base case**: A case where a recursive algorithm completes without applying itself to a smaller subproblem.

PARTICIPATION
ACTIVITY

6.10.1: Recursive factorial algorithm for positive numbers.

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Animation captions:

1. A recursive algorithm to compute N factorial has 2 parts: the base case and the non-base case.
2. N is assumed to be a positive integer. The base case is when N equals 1, wherein a result of 1 is returned.
3. The non-base case computes the result by multiplying N by (N - 1) factorial.
4. The algorithm applying itself to a smaller subproblem is what makes the algorithm recursive.

PARTICIPATION ACTIVITY

6.10.2: Recursive algorithms.



- 1) A recursive algorithm applies itself to a smaller subproblem in all cases.

 True False

- 2) The base case is what ensures that a recursive algorithm eventually terminates.

 True False

- 3) The presence of a base case is what identifies an algorithm as being recursive.

 True False

©zyBooks 06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Recursive functions

A **recursive function** is a function that calls itself. Recursive functions are commonly used to implement recursive algorithms.

Table 6.10.1: Sample recursive functions: Factorial, CumulativeSum, and ReverseList.

```
Factorial(N) {
    if (N == 1)
        return 1
    else
        return N * Factorial(N - 1)
}
```

```
CumulativeSum(N) {
    if (N == 0)
        return 0
    else
        return N + CumulativeSum(N - 1)}
```

```
ReverseList(list, startIndex, endIndex) {
    if (startIndex >= endIndex)
        return
    else {
        Swap elements at startIndex and endIndex
        ReverseList(list, startIndex + 1, endIndex - 1)
    }
}
```

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

PARTICIPATION ACTIVITY

6.10.3: CumulativeSum recursive function.



- 1) What is the condition for the base case in the CumulativeSum function?

 N equals 0

- N does not equal 0
- 2) If Factorial(6) is called, how many additional calls are made to Factorial to compute the result of 720?
- 7
 5
 3
- 3) Suppose ReverseList is called on a list of size 3, a start index of 0, and an out-of-bounds end index of 3. The base case ensures that the function still properly reverses the list.
- True
 False

@zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

6.11 Recursive algorithms



This section has been set as optional by your instructor.

Fibonacci numbers

The **Fibonacci sequence** is a numerical sequence where each term is the sum of the previous 2 terms in the sequence, except the first 2 terms, which are 0 and 1. A recursive function can be used to calculate a **Fibonacci number**: A term in the Fibonacci sequence.

Figure 6.11.1: FibonacciNumber recursive function.

```
FibonacciNumber(termIndex) {  
    if (termIndex == 0)  
        return 0  
    else if (termIndex == 1)  
        return 1  
    else  
        return FibonacciNumber(termIndex - 1) + FibonacciNumber(termIndex  
- 2)  
}
```

@zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

PARTICIPATION
ACTIVITY

6.11.1: FibonacciNumber recursive function.

- 1) What does FibonacciNumber(2) return?

/

Check

Show answer

- 2) What does FibonacciNumber(4)
return?

Check**Show answer**

- 3) What does FibonacciNumber(8)
return?

Check**Show answer**

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

Recursive binary search

Binary search is an algorithm that searches a sorted list for a key by first comparing the key to the middle element in the list and recursively searching half of the remaining list so long as the key is not found.

Binary search first checks the middle element of the list. If the search key is found, the algorithm returns the index. If the search key is not found, the algorithm recursively searches the remaining left sublist (if the search key was less than the middle element) or the remaining right sublist (if the search key was greater than the middle element).

Figure 6.11.2: BinarySearch recursive algorithm.

```
BinarySearch(numbers, low, high, key) {
    if (low > high)
        return -1

    mid = (low + high) / 2
    if (numbers[mid] < key) {
        return BinarySearch(numbers, mid + 1, high,
key)
    }
    else if (numbers[mid] > key) {
        return BinarySearch(numbers, low, mid - 1,
key)
    }
    return mid
}
```

PARTICIPATION
ACTIVITY

6.11.2: Recursive binary search.

Suppose BinarySearch(numbers, 0, 6, 42) is used to search the list (14, 26, 42, 59, 71, 88, 92) for key 42.

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023

- 1) What is the first middle element that is compared against 42?

- 42
- 59
- 71



2) What will the low and high argument values be for the first recursive call?

- low = 0
high = 2
- low = 0
high = 3
- low = 4
high = 6

3) How many calls to BinarySearch will be made by the time 42 is found?

- 2
- 3
- 4

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023



PARTICIPATION ACTIVITY

6.11.3: Recursive binary search base case.



1) Which does not describe a base case for BinarySearch?

- The low argument is greater than the high argument.
- The list element at index `mid` equals the key.
- The list element at index `mid` is less than the key.

©zyBooks 06/06/23 23:52 1692462
Taylor Larrechea
COLORADOCSPB2270Summer2023