# Activity #27: Recursion
## Recorder's Report

Manager: Reader:

Recorder: Driver:

Date: Score: Satisfactory / Not Satisfactory

Record your team's answers to the key questions (marked with  ) below.

a) Model 1, Question #2

b) Model 2, Question #6

c) Model 3, Question #15

# Activity #27: Recursion

In this activity, you will work in teams of 3–4 students to learn new concepts. This activity will introduce you to recursion in C++.

## Content Learning Objectives

*After completing this activity, students should be able to:*

- Identify the base case and recursive step of the factorial function
- Trace a recursive function by hand and and predict its final output
- Explain what happens in memory when a function calls itself

## Process Skill Goals

*During the activity, students should make progress toward:*

- Write a recursive function to compute the sum of the first $n$ numbers

# Model 1 The Factorial Function

| $n$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $n!$ | 1 | 1 | 2 | 6 | 24 | 120 |

*Refer to Model 1 above as your team develops consensus answers to the questions below.*

## Questions (15 min) Start time:

**1**. In mathematics, the *factorial* function for a natural number $n$ is denoted by $n!$. It is the product of all positive integers less than or equal to $n$. For example:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Consider how to calculate 4!.

a) Write out all of the numbers that need to be multiplied to get 4!.

b) Rewrite the expression using 3! instead of $3 \times 2 \times 1$.

**2**. Express the factorials as a product of a single natural number with a simpler factorial.

a) $3! =$

b) $2! =$

c) $100! =$

d) $n! =$

Now consider the very first natural number, 0.

a) Based on the model, what is the value of 0!?

b) Does it make sense to define 0! in terms of a simpler factorial? Explain.

c) When we define the value of a function by referencing that same function for a simpler value, we will eventually reach a point where there are no simpler values and we have to just give a concrete value to the function. This is called a *base case*. What is the base case for the factorial function?

**3**. Suppose you already have a working implementation of the function declared below.

```
int factorial(int n);
```

a) How could you compute 100! without calling `factorial(100)`? Give a C++ command to do this.

b) How could you compute $n!$ without calling `factorial(n)`? Give a C++ command to do this.

## Model 2 A C++ Factorial Function

```
1  int factorial(int n) {
2    cout << "n is " << n << endl;
3    if (n == 0) {
4      return 1;     // base case
5    } else {
6      cout << "need factorial of " << (n-1) << endl;
7      int answer = factorial(n-1);
8      cout << "factorial of " << (n-1) << " is " << answer << endl;
9      return n * answer;
10   }
11 }
12
```

*Refer to Model 2 above as your team develops consensus answers to the questions below.*

### Questions (15 min) Start time:

**4.** This model gives a definition of the `factorial` function. Use it to answer the following questions.

a) What specific function is called on line 10?

b) Why is the `if` statement on line 6 needed?

**5.** A function that calls itself is called *recursive*. What two steps are required to define the recursive function
factorial?

**6.** Because recursive functions call themselves as a part of their execution, it takes some  thought to understand their execution.

a) How many distinct function calls would be made to the `factorial` function to compute 2!? Identify the function argument for each of those calls.

b) How many distinct function calls would be made to the `factorial` function to compute 4!? Identify the function argument for each of those calls.

**7**. The file `activity27a.cpp.cpp` contains the function from this model along with a test function call to compute 5!. Run this program and then identify the function call which produces each line of output below. Several have been done for you.

a) `n is 5` `factorial(5)`

b) `need factorial of 4`

c) `n is 4`

d) `need factorial of 3`

e) `n is 3`

f) `need factorial of 2`

g) `n is 2`

h) `need factorial of 1` `factorial(2)`

i) `n is 1`

j) `need factorial of 0`

k) `n is 0`

l) `factorial of 0 is 1` `factorial(1)`

m) `factorial of 1 is 1`

n) `factorial of 2 is 2`

o) `factorial of 3 is 6`

p) `factorial of 4 is 24`

**8**. What happens if you try to calculate the factorial of a negative number? Explain why this happens.

**9**. How could you prevent this behavior?

**10**. What is the largest factorial you can compute in C++ without changing the types of the variables in this function? Play with the code in `activity08.cpp` to find out.

# Model 3 Summations

$$\sum_{i=1}^{100} i = 1 + 2 + 3 + \cdots + 100 = 5050$$

*Refer to Model 3 above as your group develops consensus answers to the questions below.*

## Questions (20 min) Start time:

**11**. In mathematics, *summation* (represented by the Greek letter "sigma", $\Sigma$) is the addition of a sequence of numbers resulting in a single sum or total. For example,

$$\sum_{i=1}^{i=3} i = 1 + 2 + 3 = 6$$

Consider how to calculate $\sum_{i=1}^{5} i$.

a) Write out all the numbers that need to be added.

b) Show how this sum can be calculated in terms of a smaller summation.

**12**. Express the summations as a sum of a single natural number and a shorter summation.

a) $\sum_{i=1}^{100} i =$

b) $\sum_{i=1}^{n} i =$

c) The base case for this summation is:

**13**. Write a C++ function `summation` that takes a single parameter `n` and returns the sum $1 + 2 + \cdots + n$. It should only have an `if` statement and two `return` statements.

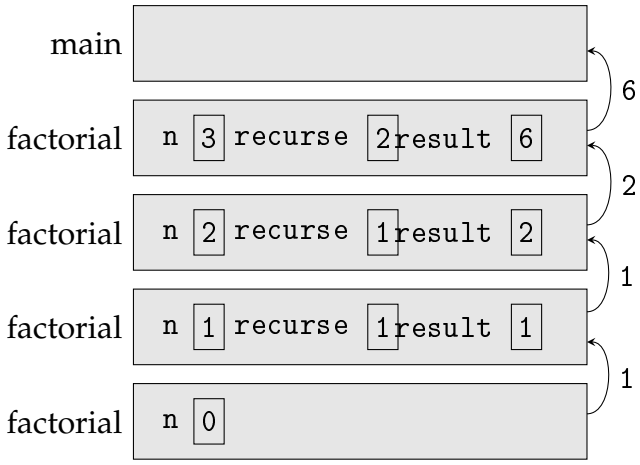
**14.** Below is a different recursive implementation of the `factorial` function seen in model 2.

```
1  int factorial(int n) {
2    if (n == 0) {
3      return 1; // base case
4    }
5    int recurse = factorial(n-1);
6    int result = n * recurse;
7    return result;
8  }
9
```

a) How are temporary variables used in this function?

b) What would you change to change this to a `summation` function?

**15.** Below is a *stack diagram* of a call to this implementation of `factorial(3)` from the `main` program. Sketch a similar diagram for a call to `summation(3)`.




a) Why are there no values for `recurse` and `result` in the stack diagram for the last call to `factorial` (when `n == 0`?

b) Looking at the stack diagram, how is it possible that the parameter `n` can have multiple values in memory at the same time?