Learning Objectives: Returning Values

- **Use the `return` keyword to return a value**
- **Identify the return value of the print statement**
- **Demonstrate the ability to return several different data types**
- **Create and apply helper functions**

Returning Values

The Return Keyword

Instead of just **printing** data, functions can also **return** data. Think of the `sizeof()` and `length()` functions. They help return the size or length (in integer) of an array and string respectively. So the return value of these functions is of type `int`. Both `sizeof()` and `length()` do not print anything to the screen, they just return a number. From here on out, user-defined functions will avoid just printing to the screen. Instead, they will return a value. To return a value, simply use the `return` keyword.

```
/**
 * This function adds 5 to an integer
 *
 * @param num An integer
 * @return The integer added to 5
 */
int AddFive(int num) {
  return num + 5;
}

int main() {
  AddFive(10);
  return 0;
}
```

You'll notice the program no longer prints anything to the screen, which is the cause for the message, `Command was successfully executed.`. This happens because the function only adds `5` to whatever parameter is passed to the function and then returns it *internally*. To *see* the result, explicitly tell C++ to print the return value of the function to the screen.

```cpp
/**
 * This function adds adds 5 to an integer
 *
 * @param num An integer
 * @return The integer added to 5
 */
int AddFive(int num) {
  return num + 5;
}

int main() {
  int newNum = AddFive(10);
  cout << newNum << endl;
  return 0;
}
```

→15

challenge

What happens if you:

- Remove all lines of code within `main()` and replace them with just `cout << AddFive(10) << endl;` and then `return 0;`? →15

Returning Values

Functions can return any value in C++ — ints, doubles, strings, vectors, etc.

```
/**
 * This function adds two integers together
 *
 * @param x The first integer
 * @param y The second integer
 * @return x added to y
 */
int ReturnInt(int x, int y) { //int function
  return(x + y);
}

/**
 * This function adds two doubles together
 *
 * @param x The first double
 * @param y The second double
 * @return x added to y
 */
double ReturnDouble(double x, double y) { //double function
  return(x + y);
}

/**
 * This function adds two strings together
 *
 * @param x The first string
 * @param y The second string
 * @return x added to y
 */
string ReturnString(string x, string y) { //string function
  return(x + y);
}

int main() { //int function
  cout << ReturnInt(1, 2) <<endl;
  cout << ReturnDouble(1, 2) <<endl;
  cout << ReturnString("1", "2") << endl;
  return 0;
}
```

challenge

Can you write a function that returns a vector?

If you want to return a vector, one possible approach is to have a vector be passed as a parameter. You can then modify the vector in some way, and then return it to the system.

▼ **Sample Code**

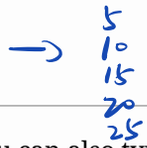
The code below takes a vector of numbers as a parameter for the function `MultiplyFive()`. The function creates a new empty vector, multiplies each element of the parameter vector by 5, and then adds those new products to the new vector. Finally, the new vector is returned. To print the returned vector, use another enhanced `for` loop to iterate through the vector after it has been initialized.

```
vector<int> MultiplyFive(vector<int>& my_list) {
  vector<int> new_list;
  for (auto a : my_list) {
    new_list.push_back(a * 5);
  }
  return new_list;
}

int main() {
  vector<int> numbers;
  numbers.push_back(1);
  numbers.push_back(2);
  numbers.push_back(3);
  numbers.push_back(4);
  numbers.push_back(5);

  vector<int> print_list = MultiplyFive(numbers);
  for (auto a : print_list) {
    cout << a << endl;
  }

  return 0;
}
```



Alternatively, you can also type the function as `void` which results in fewer lines of code.

```
void MultiplyFive(vector<int>& my_list) {
  vector<int> new_list;
  for (auto a : my_list) {
    new_list.push_back(a * 5);
  }
  for (auto a : new_list) {
    cout << a << endl;
  }
}

int main() {
  vector<int> numbers;
  numbers.push_back(1);
  numbers.push_back(2);
  numbers.push_back(3);
  numbers.push_back(4);
  numbers.push_back(5);

  MultiplyFive(numbers);
  return 0;
}
```

Helper Functions

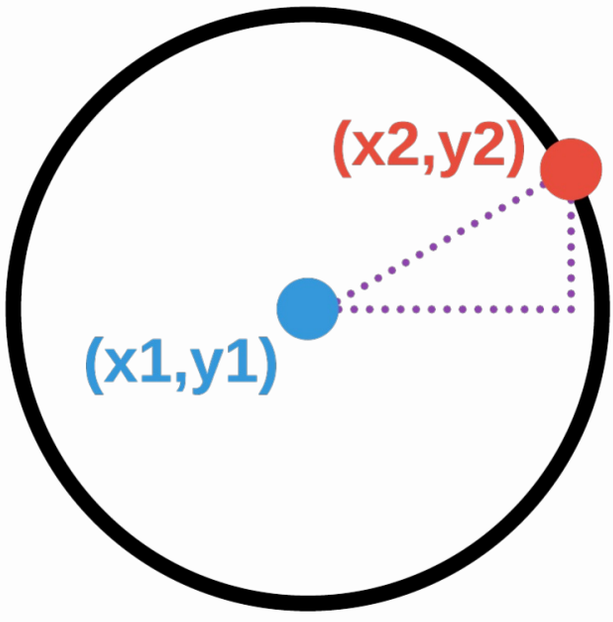
Helper Functions

Helper functions are functions that are called from within other functions. Take, for example, the formula for calculating the area of a circle:

$$A = \pi r^2$$

It would be quite easy to write a C++ function to calculate the area of a circle. However, instead of knowing the radius of the circle, you have the X/Y coordinates for a point at the center of the circle and another point on the circle. The distance formula (which is based on the Pythagorean Theorem) can calculate the radius of the circle.

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$



.guides/img/Radius

The `FindRadius()` function uses the distance formula to calculate the distance between 2 pairs of points. The `FindArea()` function finds the area of a circle by relying on the `FindRadius()` function. Therefore, the

`FindRadius()` function is a helper function. Helper functions help shorten how much code is needed to accomplish certain tasks.

```
/**
 * This function finds the radius of a circle given
 * two coordinate points
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return The radius of a circle in double
 */
double FindRadius(double x1, double y1, double x2, double y2) {
  return(sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2)));
}

/**
 * This function finds the area of a circle given
 * two coordinate points
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return The area of a circle in double
 */
double FindArea(double x1, double y1, double x2, double y2) {
  return(M_PI * pow(FindRadius(x1, y1, x2, y2), 2));
}

int main() {
  cout << FindArea(0.0, 0.0, 4.0, 4.0) << endl;
  return 0;
}
```

→ 100.531

info

Math Functions

Note that in order to perform certain functions such as finding a square or an exponent, we need to include `<math.h>` in our header as well as to define `M_PI` which represents `pi`. If you remove these from the program header, the math functions associated with `<math.h>` will no longer work. In essence, these functions also serve as helper functions.