To complete a task in a program we can use a function. Whenever a function is used in a program, we must make sure its three components are in that program. They are:

- Prototype
- Definition
- Call

You should generally start with a call of a function. For example, if the task is to compute the largest number of two integers, we can make a call of that function as:

```cpp
int x, y, Largest;
Largest = FindLargest(x, y);
```

You can see there are three parts (red, green, black) in that call. The three parts define the three parts in a header of that function: return data type, name and parameter list. Header of the function is:

```cpp
int FindLargest(int a, int b)
```

The prototype is the same as the header with a semicolon at the end:

```cpp
int FindLargest(int a, int b);
```

In a prototype, it is not necessary to include the names of the variables. (It is not a mistake to include them). So, the prototype is

```cpp
int FindLargest(int, int);
```

The only part that is missing is the body of the function. A program that reads two integers and uses a function to find the largest integer is as follows:

```cpp
int FindLargest(int, int);  // Prototype
main()
{
    int x, y, Largest;
    cout << “Enter two integers: ”;
    cin >> x >> y;
    Largest = FindLargest(x, y);  // Call
cout << “The largest integer is: ” << Largest << endl;
    return 0;
}
```

// Definition has two parts: Header and a Body
```cpp
int FindLargest(int a, int b)  // Header
{
    if (x > y) return x;  // Body
    else return y;
}
```

**Example:** Given the following declarations and call of functions, write headers and prototypes of the functions.

<table>
<thead>
<tr>
<th>Call</th>
<th>Header</th>
<th>Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>z = f(x, y);</td>
<td>float f(int a, char b)</td>
<td>float f(int, char);</td>
</tr>
<tr>
<td>print(p, 10);</td>
<td>void print(int *a, int b)</td>
<td>void print(int *, int);</td>
</tr>
<tr>
<td>y = h(z, x);</td>
<td>char h(float a, int b)</td>
<td>char h(float, int);</td>
</tr>
</tbody>
</table>
Passing Parameters in a Function

You can pass a parameter in a function call in three different ways:

- **By Value**
- **By Name**
- **By Reference**

If you want the function to change the value of the variable, send it either by name or by reference. If you do not want the function to change the value of a variable send it by value. The syntax is as follows:

<table>
<thead>
<tr>
<th>Call</th>
<th>By Value</th>
<th>By Name</th>
<th>By Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>You are sending</td>
<td>the value in x</td>
<td>the name of the variable x</td>
<td>the address of the variable x</td>
</tr>
<tr>
<td>Header</td>
<td>void f(int a)</td>
<td>void f(int &amp; a)</td>
<td>void f(int *a)</td>
</tr>
<tr>
<td>In the function environment</td>
<td>Creates a new memory location for the local variable a, and the value sent is stored in a</td>
<td>a is the name for the variable x in the function environment</td>
<td>Creates a new memory location for a, and the address of x is stored in it. Hence x gets another name and it is *a.</td>
</tr>
</tbody>
</table>

**Example:** Do a manual trace of the function call, and write the exact output.

```cpp
int F(int, int *, int &); // Prototype
main()
{
    int x=2, y=4, z=5;
    x = F(x, &y, z); // Call
    cout << x << y << z << endl;
    return 0;
}
```

```cpp
int F(int a, int *b, int &c)
{
    a = a + c;
    c = *b + a;
    *b = 2 * a;    // <---
    return c;
}
```

The exact output is **111411**

**Note:**
- In the example to find the largest integer, the function `FindLargest` sends both parameters by value.
- The prototype of a function is required by the compiler to complete type checking. If the Definition of a function is written prior to a Call of a function then its Prototype is not required.
How To Use Functions

Let us take a simple problem and see how functions can be used to solve that problem.

**Problem:** Write a program that

- Reads 10 integers and stores them in an array.
- Computes the largest
- Computes the smallest
- Computes the average
- Prints the array
- Prints all three computed values.

Each task listed above must be completed using a function.

Let us declare the variables that we need to store values:

- To store 10 integers: `int x[10];`
- To store the largest and the smallest: `int largest, smallest;`
- To store the average: `float average;`

To write the code, we will start with a main program, where each task will be completed by a call of a function. To write a call of a function, we need 3 parts:

- **Name of the function:** Give a name that describes the task it has to complete.
- **Return data type:** Is the function going to return a value? If so, write the call as `x = Name(...),` where `x` is a variable that stores the value the function returns. Otherwise write the call as `Name(...).
- **List of variables:** Determine the variables to send to the function so that the function can complete the task. For each variable, determine if you want the function to change its value. If the value should not be changed send it by value otherwise send it either by name or by reference.

Now, we will go through each task and determine the call.

- **Reads 10 integers and stores them in an array.**
  - Name: Read.
  - Not returning a value, so the return data type is void.
  - Function needs the array and the number of integers to read. We will send the array by reference: that is, send the address of the first location. We will send the number of integers 10, by value.
  - Hence the call is `Read(x, 10);`

- **Computes the largest**
  - Name: FindLargest.
  - Not returning a value, so the return data type is void.
  - Function needs the array, the number of integers to process, and a location to store the largest integer. We will send the array by reference: that is, send the address of the first location. We will send the number of integers 10, by value. Since the value in `largest` must be changed by the function, we will send it by reference.
  - Hence the call is `FindLargest(x, 10, &largest);`

- **Computes the smallest**
  - Name: FindSmallest.
  - Not returning a value, so the return data type is void.
  - Function needs the array, the number of integers to process, and a location to store the smallest integer. We will send the array by reference: that is, send the address of the first location. We will send the number of integers 10, by value. Since the value in `smallest` must be changed by the function, we will send it by name.
  - Hence the call is `FindSmallest(x, 10, smallest);`
• Computes the average
  o Name: FindAverage.
  o Let the function return the average. Hence the return data type is float.
  o Function needs the array, the number of integers to process. Since it is returning the average, it does not need a variable to be sent in the parameter list. We will send the array by reference: that is, send the address of the first location. We will send the number of integers 10, by value.
  o Hence the call is average = FindAverage(x, 10);

• Prints the array
  o Name: Print.
  o Not returning a value, so the return data type is void.
  o Function needs the array and the number of integers to print.
  o Hence the call is Print(x, 10);

• Prints all three computed values.
  o Name: Print. Note that we are using the same name (this is overloading the function Print) but the signature of this Print is different from the previous one. Signature of a function is determined by its name along with the data types of its parameters.
  o Not returning a value, so the return data type is void.
  o Function needs all three values. Their values should not be changed by the function, hence all must be sent by value.
  o Hence the call is Print(largest, smallest, average);

Now to implement each function we need to write its definition. A definition has two parts: Header and Body. A header has 3 parts corresponding to the 3 parts in a call. As you finish the header of a function, make sure to write its prototype (copy and paste the header followed by a semicolon) on top of the main function.

• Call: Read(x, 10);
  Since x has the address of the first location of the array, it is a pointer to an integer. Hence its data type is int *.
  • Header: void Read(int *a, int n)

• Call: FindLargest(x, 10, &largest);
  Since x has the address of the first location of the array, it is a pointer to an integer. Hence its data type is int *.
  The third parameter has an address of an integer variable, hence its data type is int *.
  • Header: void FindLargest(int *a, int n, int *max)

• Call: FindSmallest(x, 10, smallest);
  Since x has the address of the first location of the array, it is a pointer to an integer. Hence its data type is int *.
  The third parameter is sent by name. Hence its data type is int &.
  • Header: void FindSmallest(int *a, int n, int &min)

• Call: average = FindAverage(x, 10);
  Since x has the address of the first location of the array, it is a pointer to an integer. Hence its data type is int *.
  Since the function returns a value of type float. The return data type is float.
  • Header: float FindAverage(int *a, int n)

• Call: Print(x, 10);
  Since x has the address of the first location of the array, it is a pointer to an integer. Hence its data type is int *.
  • Header: void Print(int *a, int n)

• Call: Print(largest, smallest, average);
  Since x has the address of the first location of the array, it is a pointer to an integer. Hence its data type is int *.
  • Header: void Print(int max, int min, float avg)

Note: Remember, this is just an example to show how functions can be used in a program. The example illustrates how to write headers when a function has a variable that is sent by value, by reference or by name. You also saw how to write a call and the header of a function that returns a value. For such a simple problem, you do not have to use functions. Moreover, it would be efficient if we had used one function, instead of three functions, to compute all 3 values. As an exercise, and to practice writing programs with functions, for the same example, write a program that uses different kinds of functions: functions that would combine some of the tasks listed in the problem.
Now, let us put all the pieces together, including the body for each function, and write the complete program:

```c
// Prototypes
void Read(int *a, int n)
void FindLargest(int *a, int n, int *max)
void FindSmallest(int *a, int n, int &min)
float FindAverage(int *a, int n)
void Print(int *a, int n)
void Print(int max, int min, float avg)

main()
{
    int x[10], largest, smallest;
    float average;
    // Calls
    Read(x, 10);
    FindLargest(x, 10, &largest);
    FindSmallest(x, 10, smallest);
    Average = FindAverage(x, 10);
    Print(x, 10);
    Print(largest, smallest, average);
    return 0;
}

// Definitions
void Read(int *a, int n)
{
    for(int k=0; k<n; k++)
    {
        cin << "Enter an integer: ";
        cin >> a[k];
    }
}

void Print(int *a, int n)
{
    for(int k=0; k<n; k++)
    {
        cout << a[k] << " ";
    }
    cout << endl;
}

void FindLargest(int *a, int n, int *max)
{
    *max = a[0];
    for(int k=1; k<n; k++)
    {
        if (*max < a[k])
            *max = a[k];
    }
}

void FindSmallest(int *a, int n, int &min)
{
    min = a[0];
    for(int k=1; k<n; k++)
    {
        if (min > a[k])
            min = a[k];
    }
}

float FindAverage(int *a, int n)
{
    int sum = 0;
    for(int k=1; k<n; k++)
        sum = sum + a[k];
    return (float) sum/n;
}

void Print(int max, int min, float avg)
{
    cout << "The largest is " << max << endl;
    cout << "The smallest is " << min << endl;
    cout << "The average is " << avg << endl;
}
```

Note: See how the variables sent by reference and by name are used in the function environment.