NOTE: These framework pages are intended to assist you in taking class notes by providing a focus on important concepts and terminology. You are strongly encouraged to take additional notes on the backs of these pages.

Modern software is extremely complex, involving many components and features. A single software package such as Adobe Photoshop can involve thousands of separate programs and data files that work in unison. Such massive software systems are not developed by one programmer, but rather by teams of many analysts and programmers, each working on one small part of the larger system. Large software systems are developed by first focusing on the "big picture" (objectives) and then breaking that down into deeper and deeper levels of detail. This approach to software development is referred to as "_______________________ Design".

Most programming languages provide the ability to sub-divide a program into reusable units that can be developed as independently named blocks of code known as (depending on the language): modules, procedures, sub-routines, or (in C++, as) _________________________. These units are analyzed, designed, and coded independently and then combined to produce a complete software package.

The primary reasons for using functions in a program are to:

- allow for undetermined details to be replaced with a simple identifier until such time as the functions details can be determined. This approach is referred to as "procedural __________________________ ".
- minimize the "duplication programming steps" - known as _________________________.
- invest in the future by identifying blocks of code that could be useful in other programs and saving their documentation and code in repositories of other useful functions known as _________________________.

When designing a program with functions, an analyst should start by developing a diagram that states the name and purpose of each planned function and shows the hierarchical relationship between them. Such diagrams are called _____________________________.

Every C++ program has (at least) a function named ___________ that acts as a starting point for the program and launches (activates) other functions that might be needed to do the work of the program. The act of launching another function is referred to as ___________ that function. A function that calls another function is referred to as its ______________. A function that is called by a parent function is referred to as that parent's __________.

Each program function acts as an independent worker performing a specific task essential to the larger program. Each function must be declared within the source code before it can be called. A simple example of a function declaration might look like:

```cpp
void Display_Title() // This part of the code is the function's "header"
{
    // This part of the code (inside the braces) is the function's "body"
    cout << "Sample Program\n\n";
}
```

The source code for a function begins with its "header" which declares the data type (in this example: void) of the function's identifier (in this example: Display_Title). The parentheses following the function's identifier are always required, even if empty. The purpose of the parentheses will be described below. The header is followed by a braced block of statements known as the function's "body" which define the action(s) performed by the function. IMPORTANT: Do not type a semicolon following the header, as a proper function definition is seen as a single, large statement which includes both the header and the body. The body (block inside the braces) can contain multiple statements, each terminated by a semicolon (;).
A proper "calling" statement for the function `Display_Title()` might look like:

```
Display_Title();   // This statement calls a child function named Display_Title
```

Notice that the function name in the calling statement above must be typed exactly as it was in the declaration, including the use of upper and lower case. Function names are chosen using the same rules as variables and cannot duplicate a name already used by a variable or symbolic or named constant.

When coding a program that contains multiple functions, the code declaring a child function typically must be positioned ________________ the statement that is calling that child in the parent function. The code for a child function may follow the statement that calls it only if an extra statement containing a copy of the child function's header (only) has been written ahead of the calling statement and terminated with a semicolon. This extra statement is referred to as a ________________ ________________ statement. It does not include the statements defining the function's body (actions), as those must still be coded (following a copy of the header) in a separate function definition statement.

_____________________________________________________________________

PRACTICE:

In the space provided below, write the C++ source code for a function that will simply display the phrase "Hello world!" and a carriage return. The function's name should be `Display_Title`. Its data type should be `void`.

```

```

Next, write the C++ source code that might appear with the main parent function to call the function named `Display_Title`.

```

```

Finally, write the C++ source code for a function prototype statement that might appear ahead of the main function to allow you to position the code defining the function named `Display_Title` below the main function.

```

Some (but not all) functions need to receive data from their parent function to perform their task. Some (but not all) functions are designed to "return" data to their parent function when done with their task. Some (but not all) functions neither receive nor return any data with their parent - they just perform some steps. Data that is passed between functions is referred to as ______________________. An item of data that is received in a child function is referred to as a _______________ and must be documented in a table (similar to a Variable List) referred to as a _______________ .

The statement in a parent function that calls a child function must contain the name of the child function followed by a set of parentheses (). If the parent function passes data into the child function, the parentheses in the calling statement must contain a list of the data being passed. The item(s) in this list are referred to as "arguments" or _______________.

For example, if we wanted to write a function named CalcAvg that could receive three integer test scores and return the average of them as a floating point value, the C++ source code for that task could be written as:

```cpp
float CalcAvg (int S1, int S2, int S3) // declare function name and parameters
{
  float AVG;              // declare a "local variable"
  AVG = (S1+S2+S3) / 3;   // perform the function task
  return AVG;             // pass result back through the function name
}
```

The statement above defines a function that can calculate and return a floating point value identified by its name (CalcAvg). In a function header, all data types must be specified individually (even if they are the same) ahead of all identifiers (the function name and formal parameters). Because the variable named AVG is declared within the braced function body (block), it is a "__________ variable", meaning that use of its identifier will be valid only within the block. It will not be recognized outside it. The range of source code over which an identifier is valid is referred to as that identifier's _______________.

The proper calling statement in the main function for the CalcAvg function is shown below in boldface:

```cpp
int main () // declare the main (starting) function
{
  int SCORE1, SCORE2, SCORE3; // declare storage for 3 integer scores
  float AVERAGE;
  cout << "Enter three integer scores separated by spaces: ";
  cin >> SCORE1 >> SCORE2 >> SCORE3;
  AVERAGE = CalcAvg (SCORE1, SCORE2, SCORE3);
  cout << "The average of the three scores is: " << AVERAGE << endl;
  return 0; // return a "no error" value to the operating system
}
```

Notice the following important points regarding the declaration and calling source code of the function above:
1. No data types are mentioned in the calling statement (in main) - because it is using rather than declaring the identifiers involved, all of which have been declared already.
2. Unlike the previous example of the Display_Title function which treat its name like a command, the function name CalcAvg is being assigned like a piece of data. This is because a floating point piece of data (an average) is being returned through that function name when the function is called.
3. The formal parameter identifiers (S1, S2, and S3) do not have to match the actual parameter names (SCORE1, SCORE2, and SCORE3). The fact that they appear in the same positions in the two parameter lists is what makes them match them up (ie. SCORE1 passes to S1, SCORE2 passes to S2, etc.)
We could have positioned the code defining the `Display_Title` function beneath the `main` function, but that would have required us to add a function prototype statement above the `main` function - similar to:

```c
float CalcAvg (int S1, int S2, int S3); // declares function identifiers only
```
or just:

```c
float CalcAvg (int, int, int); // declare function name and parameter data types
```

Function prototype statements do not have to list the identifiers of the formal parameters; however they must list the data types of each parameter separately in the order they appear in the function definition header.

---

**PRACTICE:**

Examine the program code below. The program's purpose is to request and store the dollar price per share of a stock and then calculate and display the overall cost to purchase a dozen shares of it.

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

int main () // declare the main (starting) function
{
    // Declare main variables and named constants:
    const int SHARES = 12; // fixed quantity of stock shares to purchase
    double PRICE; // unit purchase price per share
    double COST; // overall cost of stock purchase

    // Introduce the program to the user:
    cout << "Fixed Lot Stock Purchase Calculator\n\n";
    cout << "This program calculates the overall cost of purchasing ";
    cout << SHARES << " shares of a single stock.\n\n";

    // Acquire input data:
    cout << "What is the dollar price per share? ";
    cin >> PRICE;

    // Calculate the overall cost
    COST = SHARES * PRICE;

    // Display and identify the cost rounded to cents
    cout << "The overall cost to purchase those " << SHARES << " shares is: ";
    cout << setprecision(2) << fixed << COST << " dollars\n";
    return 0; // return a "no error" value to the operating system
}
```

On the next page, re-code the program using 5 functions: the parent (main), and four child function:

- **Intro** - to display the program title and introduction.
- **GetPrice** - to request and return (to main) the per share stock price.
- **CalcCost** - to calculate and return the overall purchase cost using a passed in quantity and unit share price.
- **ShowCost** - to display and identify the passed in cost rounded to cents.

In all of the child functions, use easily remembered, one letter identifiers for the needed formal parameters and local variable names. Pay careful attention to data types. Some of the blanks might not apply and remain empty.
#include <iostream>
#include <iomanip>
using namespace std;

/*  ============  CHILD FUNCTIONS  ============  */

________ Intro ( _________________________________ )
{
  // Introduce the program to the user:
  cout << "Fixed Lot Stock Purchase Calculator\n\n";
  cout << "This program calculates the overall cost of purchasing ";
  cout << S << " shares of a single stock.\n\n";

}

________ GetPrice ( _________________________________ )
{
  // Acquire input data:
  cout << "What is the dollar price per share? ";
  cin >> _____;
}

________ CalcCost ( _________________________________ )
{
  // Calculate the overall cost
  C = S * P;

}

________ ShowCost ( _________________________________ )
{
  // Display and identify the cost rounded to cents
  cout << "The overall cost to purchase those " << S << " shares is: ";
  cout << setprecision(2) << fixed << C << " dollars\n";
}

/*  ============  MAIN FUNCTION  ============  */
int main () // declare the main (starting) function
{
  // Declare main variables:
  const int SHARES = 12; // fixed quantity of stock shares to purchase
  double PRICE; // unit purchase price per share
  double COST; // overall cost of stock purchase

  /* Call the necessary functions in the order needed */
  ____________ // Introduce the program to the user
  ____________ // Acquire unit stock price from keyboard
  ____________ // Calculate the overall cost
  ____________ // Display and identify the cost rounded to cents
  return 0; // return a "no error" value to the operating system
}
As shown above, the act of returning a single value from a function is easily accomplished using a return statement which copies the value in its argument into the name of the function. The functions names in the previous example were defined outside of other functions (similar to main). Function names declared outside of other functions can be referred to in any other function, making the scope of their identifiers _______________. Identifiers declared inside a function has a ________________ scope. So it cannot be used outside of that function. For example, the local variable P obtained within the GetPrice () function cannot be used outside of that function. That is why the return statement was used to copy its value into the global function name GetPrice, which is valid to use in the main function.

HOW TO RETURN MORE THAN ONE VALUE FROM A FUNCTION

The return statement cannot be used to pass back more than one value to a parent function because a child function has only one name to hold the returned data. So a different approach must be used. One way to accomplish this is to use a special type of formal parameter that behaves differently than the ones we used in the previous example. The formal parameters there receive values from the actual parameters listed in the calling statements for those functions. For example, consider a calling statement such as:

```
DisplayAnswer (ANSWER);
```

which sends a copy of its value (in this example, just the variable ANSWER) within its parenthesized actual parameter list to the corresponding formal parameter (in this example, the parameter ANS) listed in the function header of the DisplayAnswer function:

```
void DisplayAnswer (int ANS)
```

This type of data passage is referred to as "passing by ____________.". The value in ANSWER is copied into ANS, which is an extra local storage location that will exist only while the child function is running. Any type of data passage in which a value is copied from one parameter to another is referred to as "passage by value".

But there are other types of data passage. Instead of passing a value, we can pass a "reference" to a storage location. That reference can then be treated as if it was an alias for the actual parameter, but no extra storage is used. For example, consider a different function header for the DisplayAnswer function:

```
void DisplayAnswer (int &ANS)
```

Notice the addition of the symbol & ("ampersand") to the data type resulting in int& rather than just int. This type of parameter is referred to as a "______________ variable", meaning that is does not define a new storage location - just another way of referring to the actual parameter (ANSWER) in the calling statement. This type of data passage is referred to as "passing by ____________.". Note that the & symbol is part of the name of the data type rather than the parameter. In C++, the function header above would often be written as:

```
void DisplayAnswer (int &ANS)
```

Note the position of the & symbol. The change has no effect on the meaning. It still applies to the data type rather than the parameter. The parameter name would be just ANS, not &ANS.

Using this approach, we can pass data in either direction (into or out of a function). For example, we could display the value in the parent function's actual parameter ANSWER by referring to its alias reference variable ANS in the child function DisplayAnswer with a statement such as:

```
cout << ANS << endl;
```
However, we could also pass data out of the child function back to the variable \texttt{ANSWER} in the parent function. For example, we could assign a new value of zero to the parent function's actual parameter \texttt{ANSWER} by referring to its alias reference variable \texttt{ANS} in the child function \texttt{DisplayAnswer} with a statement such as:

\begin{verbatim}
ANS = 0;
\end{verbatim}

Because \texttt{ANS} is an alias for \texttt{ANSWER}, any reference to \texttt{ANS} is also a reference to \texttt{ANSWER}. This bi-directional ability provided by reference variables allows us to pass \textit{back} as many values as desired by simply using the necessary quantity of reference variables in the formal parameter list. This eliminates the need to use a \texttt{return} statement, as the function's name need not be used to pass back the data. Therefore, the function name could be declared to have a \texttt{void} data type, as in the following header:

\begin{verbatim}
void DisplayAnswer ( int &ANS)
\end{verbatim}

The ability use reference variables to pass \textit{back} data through parameter names allows us to create functions that can return multiple values to their parent function.

PRACTICE PARAMETER PASSING BY REFERENCE:

Suppose that we wanted to modify program on Page 4 to ask the user for the quantity of shares of stock to purchase rather than setting a named constant value to 12. Examine the modified program code below for a program that was developed using only one (main) function. This program's purpose is to request and store the dollar price per share of a stock \textit{and the quantity of shares to purchase} and then calculate and display the overall cost to purchase them.

\begin{verbatim}
#include <iostream>
#include <iomanip>
using namespace std;

int main () // declare the main (starting) function
{
    // Declare main variables and named constants:
    int SHARES; // quantity of stock shares to purchase
    double PRICE; // unit purchase price per share
    double COST; // overall cost of stock purchase

    // Introduce the program to the user:
    cout << "Variable Lot Stock Purchase Calculator\n\n";
    cout << "This program calculates the overall cost of purchasing ";
    cout << "a user-specified quantity of shares of a single stock.\n\n";

    // Acquire input data:
    cout << "How many shares of stock do you want to purchase? ";
    cin >> SHARES;
    cout << "What is the dollar price per share? ";
    cin >> PRICE;

    // Calculate the overall cost
    COST = SHARES * PRICE;

    // Display and identify the cost rounded to cents
    cout << "The overall cost to purchase those " << SHARES << " shares is: ";
    cout << setprecision(2) << fixed << COST << " dollars\n";
    return 0; // return a "no error" value to the operating system
}
\end{verbatim}
On page below, re-code the program using 5 functions: the parent (main), and four child function as follows:

- **Intro** - to display the program title and introduction.
- **GetData** - to request and return (to main) the per-share stock price and the quantity of shares to purchase.
- **CalcCost** - to calculate and return the overall purchase cost using a passed in quantity and unit share price.
- **ShowCost** - to display and identify the passed in cost rounded to cents.

In all of the child functions, use easily remembered, one letter identifiers for the needed formal parameters and local variable names. Pay careful attention to data types. Some of the blanks might not apply and remain empty.

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

/*  ============  CHILD FUNCTIONS  ============  */

    Intro ( _________________________________ )
{  // Introduce the program to the user:
cout << "Variable Lot Stock Purchase Calculator\n\n";
cout << "This program calculates the overall cost of purchasing ";
cout << S << " shares of a single stock.\n\n";
}

    GetData ( _________________________________ ) // Pass by reference
{  // Acquire input data:
cout << "How many shares of stock do you want to purchase? ";
cin >> ____;
cout << "What is the dollar price per share? ";
cin >> ____;
}

    CalcCost ( _________________________________ )
{  // Calculate the overall cost
    C = S * P;    // Local variable for cost
    }    // Return the calculated cost via CalcCost

    ShowCost ( _________________________________ )
{  // Display and identify the cost rounded to cents
    cout << "The overall cost to purchase those " << S << " shares is: ";
cout << setprecision(2) << fixed << C << " dollars\n";
}

/*  ============  MAIN FUNCTION  ============  */

int main () // declare the main (starting) function
{
    // Declare main variables:
    const int SHARES = 12; // fixed quantity of stock shares to purchase
    double PRICE;  // unit purchase price per share
    double COST;   // overall cost of stock purchase
    /* Call the necessary functions in the order needed */

    Intro();
    GetData(S, PRICE);
    COST = CalcCost(S, PRICE);
    ShowCost(COST);
}
```
// Introduce the program to the user

// Get stock price and quantity from keyboard

// Calculate the overall cost

// Display and identify the cost rounded to cents

return 0; // return a "no error" value to the operating system

MORE PRACTICE

For more practice using both "passing by value" and "passing by reference", look at the structure diagram at:


Then fill in the table below without looking at the related table on that web page..

<table>
<thead>
<tr>
<th>Syntax of the Function Header Declaration</th>
<th>Calling Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Func1 ()</td>
<td></td>
</tr>
<tr>
<td>Func2 (AA)</td>
<td></td>
</tr>
<tr>
<td>Func3 (AA, BB)</td>
<td></td>
</tr>
<tr>
<td>CC = Func4 ()</td>
<td></td>
</tr>
<tr>
<td>CC = Func5 (AA)</td>
<td></td>
</tr>
<tr>
<td>CC = Func6 (AA, BB)</td>
<td></td>
</tr>
<tr>
<td>Func7 (CC, DD)</td>
<td></td>
</tr>
<tr>
<td>Func8 (CC, DD, AA)</td>
<td></td>
</tr>
<tr>
<td>Func9 (CC, DD, AA, BB)</td>
<td></td>
</tr>
</tbody>
</table>