Lab 4: Functions

Note: in the following, you may find "errors" in the given code, they are there to give you debugging practice.

Your name:	Zach Evans
Your email address:	evansz2@students.wwu.edu
Your student ID number:	W01006113

Lab Objectives

To gain experience in

- relating a function's parameter/return value interface to its purpose
- tracing the flow of a function's execution
- function naming and commenting
- recognizing when to use value and reference parameters
- determining the scope of variables
- decomposing complex tasks into simpler ones
- designing functions that solve practical problems
- programming recursive functions
- using the assert macro to specify function preconditions

R1. Functions as Black Boxes

Predictable input will result in predictable output. A function maps element(s) from a problem domain, called *parameter(s)*, into an element from a range of solutions, called the *return value*.

To treat a function as a "Black Box", one simply uses it.

For instance, here's a function to compute the volume of a cylinder, say, a beer can, given the height and diameter in millimeters.

```
/* PURPOSE:   Function to compute the volume of a cylinder
   RECEIVES:  height - the height in millimeters
              diameter - the diameter in millimeters
   RETURNS:   volume - in cubic milliliters
*/
double cylinder_volume(double height, double diameter)
{  . . .
}
```

To use this function to measure the volume of a can of Blatz Lite, just supply parameters and get its return.

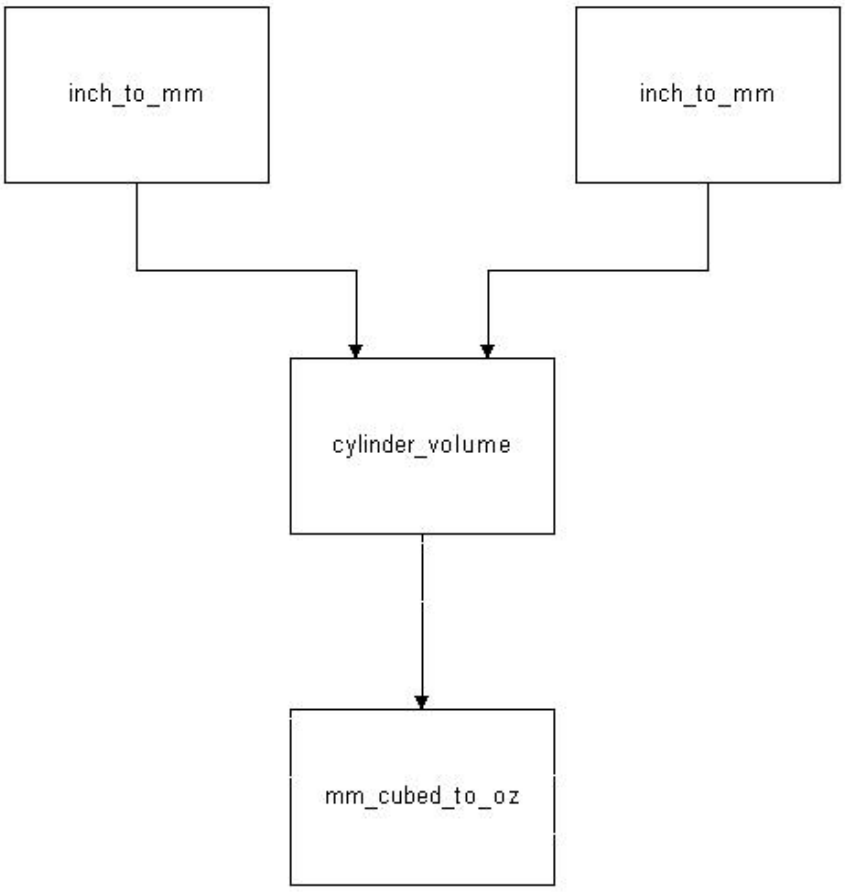
```
double v = cylinder_volume(2 * can_depth, 1.5 * can_diameter);
```

Notice that you can use the function without knowing the implementation. Actually, this function is quite

simple

```
double cylinder_volume(double height, double diameter)
{  double volume = PI * pow(diameter/2, 2) * height;
   return volume;
}
```

Suppose, instead of milliliters, an answer in fluid ounces is needed, say for a recipe. Use the following functions, together with `double cylinder_volume(double, double)` to implement a US measurement version `double cylinder_volume_oz(double, double)` like this:




```
double inch_to_mm(double inches)
/* PURPOSE:   Function to compute the volume of a cylinder
   RECEIVES:  inches - value in inches to convert to millimeters
   RETURNS:   the converted value
   REMARKS:   1 inch = 25.4 millimeters
*/
{  const double MM_PER_INCH = 25.4;
   return inches *  MM_PER_INCH;
}

double mm_cubed_to_oz(double mm3)
/* PURPOSE:   Function to convert cubic millimeters to U.S. ounces
   RECEIVES:  mm3 - volume in cubic millimeters
   RETURNS:   volume in ounces
   REMARKS:   1 fluid U.S. ounce = 29.586 milliliters
*/
```

```
{  const double MM_CUBED_PER_OZ = 29.586;
   return value_to_convert / MM_CUBED_PER_OZ;
}

double cylinder_volume(double height, double diameter)
/* PURPOSE:   Function to compute the volume of a cylinder
   RECEIVES:  height - the height in millimeters
              diameter - the diameter in millimeters
   RETURNS:   volume - in cubic milliliters
*/
{  double volume = PI * diameter * diameter * height / 4;
   return volume;
}



int cylinder_volume_oz(double height, double diameter)
/* PURPOSE:   Function to compute the volume of a cylinder
   RECEIVES:  height - the height in inches
              diameter - the diameter in inches
   RETURNS:   volume - in fl. oz.
*/
{
```

```
return mm_cubed_to_oz(cylinder_volume(inch_to_mm(height),
inch_to_mm(diameter)));
```

```
}

int main()
{  double height;
   double diameter;
   double volume;

   cout << "Please enter the height (in inches)" << "\n";
   cin >>  height;

   cout << "Please enter the diameter (in inches)" << "\n";
   cin >>  diameter;

   volume = cylinder_volume_oz(height, diameter);
   cout << "The volume is " << volume << "ounces" << "\n";

   return 0;
}
```

P1. Writing Functions

Productivity Hint 5.1 in the text suggests several enhancements to the future value function. Here is the function from the text.

```
double future_value(double initial_balance, double p, int nyear)
/* PURPOSE:  computes the value of an investment with compound interest
   RECEIVES: initial_balance - the initial value of the investment
```

```
            p-the interest rate as a percent
            nyear-the number of years the investment is held
   RETURNS:  the balance after nyears years
   REMARKS:  Interest in compounded monthly
*/

{  double b = initial_balance * pow(1 + p / (12 * 100), 12 * nyear);
   return b;
}

int main()
{   cout << "Please enter the initial investment: ";
    double initial_balance;
    cin >> initial_balance;
    cout << "Please enter the interest rate in percent: ";
    double rate;
    cin >> rate;
    cout << "Please enter the number of years: ";
    double nyears;
    cin >> nyears;
    double balance = future_value(initial_balance, rate, nyears);
    cout << "After " << nyears << ", the initial investment of " << initial_balance <<
       " grows to " << balance << "\n";

    return 0;
}
```

Change the `future_value` function to compute the value of the investment when there are `npayments` regular interest payments per year (instead of 12 monthly payments). Supply a `main` function that calls your changed function.

```
#include <iostream>
#include <string>
#include <math.h>
using namespace std;

double future_value(double initial_balance, double p, int nyear, int npayments)
/* PURPOSE:  computes the value of an investment with compound interest
 RECEIVES: initial_balance - the initial value of the investment
 p-the interest rate as a percent
 nyear-the number of years the investment is held
 npayments - the number of payments per year
 RETURNS:  the balance after nyears years
 REMARKS:  Interest in compounded monthly
 */

{
    double b = initial_balance * pow(1 + p / (12 * 100), npayments * nyear);
    return b;
}

int main()
{   cout << "Please enter the initial investment: ";
    double initial_balance;
    cin >> initial_balance;
    cout << "Please enter the interest rate in percent: ";
    double rate;
    cin >> rate;
    cout << "Please enter the number of years: ";
    double nyears;
    cin >> nyears;
    cout << "Please enter the number of payments per year: ";
    double npayments;
    cin >> npayments;
    double balance = future_value(initial_balance, rate, nyears, npayments);
    cout << "After " << nyears << " years, the initial investment of " <<
initial_balance <<
    " grows to " << balance << "\n";

    return 0;
}
```

What is the value of a $5100 investment after 4 years at 11 percent if interest is compounded quarterly?

$7871.90

Change the `future_value` function to accept holding an investment for a fractional number of years, `fyears`. Supply a `main` function that calls your changed function.

```
#include <iostream>
#include <string>
#include <math.h>
using namespace std;

double future_value(double initial_balance, double p, double fyears, int
npayments)
/* PURPOSE:  computes the value of an investment with compound interest
 RECEIVES: initial_balance - the initial value of the investment
 p-the interest rate as a percent
 fyears-the number of years the investment is held
 npayments - the number of times interest is payed per year
 RETURNS:  the balance after nyears years
 REMARKS:  Interest in compounded npayments times per year
 */

{
    double b = initial_balance * pow(1 + p / (12 * 100), npayments * fyears);
    return b;
}

int main()
{   cout << "Please enter the initial investment: ";
    double initial_balance;
    cin >> initial_balance;
    cout << "Please enter the interest rate in percent: ";
    double rate;
    cin >> rate;
    cout << "Please enter the number of years: ";
    double nyears;
    cin >> nyears;
    cout << "Please enter the number of payments per year: ";
    double npayments;
    cin >> npayments;
    double balance = future_value(initial_balance, rate, nyears, npayments);
    cout << "After " << nyears << " years, the initial investment of " <<
initial_balance <<
    " grows to " << balance << "\n";

    return 0;
```

What is the value of a $5100 investment after 5.5 years at 9 percent if interest is compounded weekly (52 weeks/year)? $8362.96

R2. Function Names

The following code can be used to test a text mode function's interface. It's called a test harness, because any function can be used in it, simply by replacing `foo()` with the new function call and the desired parameters and returns.

```
double foo(int n)
/* PURPOSE:  Test harness for calls to console programs
   RECEIVES: Programmer specified parameter
   RETURNS:  Programmer specified return value
   REMARKS:  Used to test parameter and return passing.
*/
{  cout << "Got to " << "foo" << " with parameter " << n << "\n";
   return n * 0.1;               /* to simulate function activity */
}
```

```
int main()
{  int give = 3;                /* any test value can be entered here */
   double get;
   get = foo(give);
   cout << "Returned from " << "foo" << " with " << get << "\n";

   return 0;
}
```

Why are the following functions badly named? Try using them in the test harness. What happened? What names would be better?

- `double 4toTheN(int n)`
  /* compute a power of 4 */

  ```
  The character '4' isn't usable in the name of a function.
  Better names would be something like double fourToTheN (int n), double fourExpN
  (int n), or double fourPowN (int n).
  ```

- `double(int max_value)`
  /* generate a random floating point number between 0 and max_value */

  ```
  This function has no name and is therefore uncallable. A good name for it would
  be double randNumWithMax(int max_value)
  ```

- `double rt(int x)`
  /* computes the square root */

  ```
  This function runs, but the name isn't very descriptive. It doesn't say what
  kind of root it is. A better name (and the one that is used in math.h) would be
  double sqrt(int x).
  ```

P3. Return Values

In functions with more complicated branching of control, one way to insure a reasonable return value is to gather together all the possibilities and issue only one `return` statement from the very end of the block statement. Rewrite the `points_of_compass` function as follows:

1. Introduce an additional variable `string direction_string`
2. Be more clever about the logic--first compute the major direction (north, east, south, west), then append an east or west if necessary
3. Return the `direction_string` from the end of the function only

```
string points_of_compass(int degrees)
/* PURPOSE:   Convert a numeric compass position to it's verbal equivalent
   RECEIVES:  degrees - the compass needle angle in degrees
   RETURNS:   the value as a compass direction ("N", "NE", ...)
*/
{  degrees = degrees % 360;
```

```
   double octant = degrees / 45.0 - 0.5;

   if (octant >= 7)
      return "North West";
   else if (octant >= 6)
      return "West";
   else if (octant >= 5)
      return "South West";
   else if (octant >= 4)
      return "South";
   else if (octant >= 3)
      return "South East";
   else if (octant >= 2)
      return "East";
   else if (octant >= 1)
      return "North East";
   else if (octant >= 0)
      return "North";
   else
      return "North West";
}


int main()
{  int degrees;

   cout << "Please enter the compass heading (in degrees): ";
   cin >> degrees;

   string direction = points_of_compass(degrees);
   cout << "You are heading " << direction << "\n";
   return 0;
}
```

```
#include <iostream>
#include <string>
#include <math.h>
using namespace std;
string points_of_compass(int degrees)
/* PURPOSE:   Convert a numeric compass position to it's verbal equivalent
 RECEIVES:  degrees - the compass needle angle in degrees
 RETURNS:   the value as a compass direction ("N", "NE", ...)
 */
{
    degrees = degrees % 360;
    double octant = degrees / 45.0 - 0.5;

    string direction_string;


    if (octant >= 7)
        direction_string = "North";
    else if (octant >= 6)
        direction_string = "West";
    else if (octant >= 3)
        direction_string = "South";
    else if (octant >= 2)
        direction_string = "East";
    else
        direction_string = "North";

    if (octant >= 7 || (octant >= 5 && octant < 6) || octant < 0)
        direction_string.append(" West");
    else if ((octant >= 3 && octant < 4) || (octant >= 1 && octant < 2))
        direction_string.append(" East");


    return direction_string;
}


int main()
{  int degrees;

    cout << "Please enter the compass heading (in degrees): ";
    cin >> degrees;

    string direction = points_of_compass(degrees);
    cout << "You are heading " << direction << "\n";
    return 0;
}
```

R4. Parameters

Consider the following functions:

```
string propercase(string z);
void swapit(int& a, double& b);
double cylinder_height(int h, double r);
```

and these variables:

```
string greeting = Hello!"
```

```
int f;
double g;
double x;
```

What is wrong with each of the following function calls ?

- `greeting = propercase(f);`

```
f is an integer and propercase(string z) takes a string.
```

- `double_swap_it(f, g)`

```
There's no function called double_swap_it.
```

- `greeting = cylinder_height(f,g)`

```
Greeting is a string and cylinder_height returns a double.
```

Side Effects, Procedures and Reference Parameters

Procedures differ from functions since they generally do not have a return value. Sometimes, a procedure may have a return value but the generation of a return value is not the main purpose of the procedure. Several conditions may cause a function to not return a value:

- nothing needs to be returned

```
void print_result(string out_string)
/* PURPOSE:   print a string to the screen
   REMARKS:   Procedure doesn't need to return confirmation of successful print
*/
{   cout << "The result is " << out_string << endl;
}
```

- more than one thing is changed, but `return` could only handle one of them:

```
void reset(double& level, Time& starttime, string& name, int& index)
/* PURPOSE:   Reset variables to default values
   RECEIVES:  level - the level to be reset
              starttime - the time to be reset
              name - the name to be reset
              index - the index to be reset
   REMARKS:   All parameters are passed by reference to enable changing them in place
*/
{  level = 0.0;
   starttime = Time();
   name = "";
   index++;
}
```

- the object to be changed is in the global scope

```
/* PURPOSE:   Set an employee's salary
   USES:      Global top_salary
```

```
      REMARKS:   top_salary keeps the maximum salary of all seen by set_salary
   */
   void set_salary(Employee employee, double new_salary)
   {  if (new_salary > top_salary)
         top_salary = new_salary;
      employee.set_salary(new_salary);
   }
```

R5. Tracing Reference Parameters

The street gambler's game of 3-Card Monte is deceptively simple. Three cards are placed face down on a table. One of the three is the Queen of Spades, and you are shown where it is. The dealer then re-arranges the cards, and asks "Where is the Queen?"

```
void swap(string& a, string& b)
{  string temp;
   temp = a;
   a = b;
   b = temp;
}

int three_card_monte(string card1, string card2, string card3)
{  swap(card1, card2);
   swap(card2, card3);
   swap(card1, card3);

   if (card1 == "queen")
      return 1;
   else if (card2 == "queen")
      return 2;
   else /* (card3 == "queen") */
      return 3;
}

int main()
{  string first = "queen";
   string second = "king";
   string third = "ace";
   int guess;
   int location;

   location = three_card_monte(first, second, third);

   cout << "Where's the Queen ( 1, 2 or 3 ) ? ";
   cin >> guess;

   if (guess == location)
      cout << "Congratulations!" << "\n";
   else
      cout << "Better luck next time, it was number " << location << "\n";

   return 0;
}
```

What are the values of the parameters to `three_card_monte(card1, card2, card3)`

```
"queen", "king",
"ace"
```

To what variable do `a` and `b` refer in the first, second and third calls to `swap(a,b)`

call	a	b
1	queen	king
2	queen	ace
3	king	queen

P4. Programming with Reference Parameters

Write a function `sort3d()` that sorts three integers in decreasing order. You may use:

```
void sort2d(int& a, int&b)
{   int temp;

    if (a < b)
    {  temp = b;
       a = b;
       b = temp;
    }
}
```

```
void sort3d(int& a, int& b, int& c)
{
    sort2d(a, c);
    sort2d(a, b);
    sort2d(b, c);
}
```

R6. Variable scoping

Generally, we want to encourage you to define a variable when you first need it, but you have to pay attention to the scope. Find what's wrong with this function's variable scoping, then fix it.

```
/* PURPOSE:   Select the maximum of three integer values
   RECEIVES:  ints i, j, k
*/

int maximum(int i, int j, int k)
{  if (i > j)
   {  int a;
      a = i;
   }
   else
   {   a = j;
   }

   if (k > a)
   {   return k;
   }
   else
   {   return a;
   }
}
```

```
int maximum(int i, int j, int k)
{
    int a;
    if (i > j)
    {
        a = i;
    }
    else
    {
        a = j;
    }

    if (k > a)
    {
        return k;
    }

    else
    {
        return a;
    }
```

P5. Eliminating Global Variables

Global variables may "work", but the advantages they offer are outweighed by the confusion they can cause. Since all functions can set a global variable, it is often difficult to find the guilty party if the global variable is set to the wrong value.

```
int maximum;


void set_max(int a)
/* PURPOSE:   Updates maximum if parameter is larger
   RECEIVES:  a - the value to compare against maximum
   REMARKS:   Uses global int maximum
*/
{  if (maximum < a)
   {   maximum = a;
   }
}
void max3(int i, int j, int k)
{  maximum = i;
   set_max(j);
   set_max(k);
   return maximum;
}

int main()
{   cout << "Please enter the first integer: ";
    cin >> i;
    cout << "Please enter the second integer: ";
    cin >> j;
    cout << "Please enter the third integer: ";
    cin >> k;
    maximum = max3(i, j, k);

    cout << "The maximum is " << maximum << "\n"
    return 0;
}
```

Re-write `max3()` to avoid the use of global variables, and to preserve the logic of the function.

```
int max3(int i, int j, int k)
{
    int max;
    max = i;
    if (max < j)
    {
        max = j;
    }

    if (max < k)
    {
        max = k;
    }
    return max;
}
```

R7. Walkthroughs

Check the arguments accepted by `three_card_monte()`. What happens if one of them is empty, e.g. `three_card_monte("queen","king","")` ?

```
It will still work because "" is still a string.
```

What changes to the function would you recommend ?

```
I would check if the arguments provided were actual card names.
```

P7. Recursion

Consider a function `int digits(int)` which finds the number of digits needed to represent an integer. For example, `digits(125)` is 3 because 125 has three digits (1, 2, and 5). The algorithm is defined as:

if n < 10, then digits(n) equals 1. Else, digits(n) equals digits(n / 10) + 1.

(Why? If n is less than 10, one digit is required. Otherwise, n requires one more digit than n/10.)

For example, if called as `int num_digits = digits(1457)`, the following trace results:

```
digits(1457)
= digits(145) + 1
= digits(14) + 1 + 1
= digits(1) + 1 + 1 + 1
= 1 + 1 + 1 + 1
```

Do a trace of `digits(32767)`

```
digits(32767)
= digits(3276) + 1
= digits(327) + 1 + 1
= digits(32) + 1 + 1 + 1
= digits(3) + 1 + 1 + 1 + 1
= 1 + 1 + 1 + 1 + 1
```

Write `int digits(int)` to be called by the following `main()`:

```
int main()
{  int test_value;

   cout << "Please enter a number " << "\n";
   cin >>  test_value;

   int ndigits = digits(test_value);
   cout << "You need " << ndigits << " bits to represent " << test_value << " in decimal\n";
   return 0;
}
```

```
int digits(int n)
{
    if (n < 10)
    {
        return 1;
    }

    else
    {
        return 1 + digits(n/10);
    }
}
```

R8. Preconditions

The following program computes the percentage difference between last year's raise and this year's raise.

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

double compute_change(double ns, double ls, double os)
{
        return (((ns-ls)/(ls-os))*100);
}

int main()
{
        double new_sal, old_sal, last_sal;
        cout << "New salary: ";
        cin >>new_sal;
        cout << "\nLast year's salary: ";
        cin >>last_sal;
        cout <<"\nPrevious year's salary: ";
        cin >> old_sal;
        cout <<compute_change(new_sal,last_sal,old_sal);
        return 0;
}
```

Rewrite the compute_change function using two different methods to insure that the denominator (last year's salary - previous year's salary) is not equal to zero.

```
double compute_change(double ns, double ls, double os)
{

    if ((ls-os) != 0)
    {
        return (((ns-ls)/(ls-os))*100);
    }

    else
    {
        cout << "Invalid denominator";
        return 0;
    }

}
```

```
double compute_change(double ns, double ls, double os)
{

    if (ls != os)
    {
        return (((ns-ls)/(ls-os))*100);
    }

    else
    {
        cout << "Invalid denominator";
        return 0;
    }

}
```

Don't forget to print this page to a pdf file when you're finished and submit the pdf file to moodle.