This worksheet is for your use during and after lecture. It will not be collected or graded, but I think you will find it a useful tool as you learn C++ and study for the exams. Explain all false answers for the "True or False" questions; in general, show enough work and provide enough explanation so that this sheet is a useful pre-exam review. I will be happy to review your answers with you during office-hours, via Email, or instant messaging.

1. True or False: `Calc%O2` is a valid C++ function name. If false, state why.

2. True or False: You have not written any user-defined functions in the course assignments thus far. If false, name the function or functions.

3. Write the function definition for a function that returns the sum of `int` and `double` arguments as a double value. Call the function `silly_sum`.

4. True or False: The compiler must encounter every function's prototype or definition before it is used in a program listing. If false, cite a counter example.

5. True or False: Every user-defined function must have a prototype *in the same source file* as its function definition. If false, cite a counter example.

6. (a) Write a function prototype for your answer to question 3.

   (b) Can you write another, equally correct, function prototype for question 3?

7. Write the function definition for a `void` function that does not accept arguments and does nothing.

   (a) Call the function `Super_Void`.

   (b) Now, write a different function definition for the same `void` function that differs by one word.

8. True or False: There are two different techniques for a programmer to show that a function does not return a value.

9. Explain in detail the difference between the two function prototypes below, and what they mean to the programmer *using* these functions.

```
int a( int i, string& s );          int b( int& i, string& s );
```

10. What is a `void` function?

11. Write the prototype for a function that returns a Boolean value and does not accept any arguments. Call the function `DecisionMaker`.

12. (a) What limitation does the return value have for a (non-`void`) function that must communicate values back to its caller?

    (b) What mechanism may be used by a function that must communicate more than one data value back to its caller?

    (c) We have learned another technique that might be used — what is it?

13. (a) Write a function called `CircleMeasures` that is called with the radius of a circle (as a `double`) and calculates the diameter, circumference, and area for the caller.

    (b) In what ways might your classmates' answers differ from yours yet still be correct?

14. Using a `while` loop, write the function definition for `factorial`, a function that returns an integer, accepts a single integer argument `x` and returns the value of $x!$. Write a simple `main` routine that tests this function.

15. Recall from Calculus II that the Taylor series expansion of $f(x) = e^x$ is

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Use your solution for calculating $x!$ above (question 14) to write the function definition for

```
double exp_taylor_term( double x, int n );
```

that returns the n^{th} term of the Taylor series for e^x.

16. Which of the following will the compiler use to distinguish functions by their signatures.

    A. Function return type

    B. `const`-ness of function return type

    C. The `const`-ness of a user-defined class' member function

    D. The user-defined class containing a member function

    E. The argument types

    F. The order of argument types

    G. The argument names

    H. The order of argument names

    I. The `const`-ness of function arguments

17. Consider the following snippet of code:

```
3  void f( int x, double y );
4  int f( const string& s, int x );
5  double f( string& s, int x );
6  double f( string& s, double y );
7
8  int main()
9  {
10     const string greeting( "hello" );
11     string farewell( "good-bye" );
12     int i( f( greeting, 3 ));
13     f( i, f( farewell, 3.0 ));
14     return 0;
15 }
```

For the following line numbers, determine which functions are called, and in what order (identify a function by its prototype's line number).

(a) Line 12.

(b) Line 13.

18. Consider the following snippet of code:

```
3  void f( int& x );
4  void f( const int& );
5  bool f( int& );
6  bool f( double y );
7  bool f( int );
8  bool f( double x );
9  void f( int x );
10 void f();
```

Find the two pairs of functions prototypes with matching signatures that will cause a compiler error.

19. Consider the code snippet below to answer the questions at the right.

```
4  int add_two( int a, int& b )
5  {
6      b -= 1;
7      a += 1;
8      return a + b;
9  }
10
11 int add_three( int a, int& b, int c )
12 {
13     return add_two( add_two( a, b ), c );
14 }
15
16 int main()
17 {
18     int a = 10;
19     int b = 20;
20     int c = 0;
21
22     cout << add_three( a, b, c ) << "␣";
23     cout << a << "␣" << b << "␣" << c;
24     cout << endl;
25
26     return 0;
27 }
```

(a) Does this snippet represent function composition? On which lines?

(b) How many times is line 8 traversed by the CPU? What are the values of `a` and `b` each time?

(c) What is printed by `main()`?

20. Below is a snippet of C++ source from a file named `oops.cxx`, and below that is an error listing generated during a build attempt. Use the space at the right to explain how the code may be changed to resolve each error message.

```
1  #include <string>
2  #include <cmath>
3  using namespace std;
4
5  int f( int a );
6
7  string g( double f );
8
9  void h( int m, string& n )
10 {
11     return;
12 }
13
14 int main()
15 {
16     double product = j();
17     string text( "abc" );
18     int result;
19     double product_string;
20
21     product_string = g( product );
22
23     result = h( f('A')+h(0,text), "xyz" );
24
25     return 0;
26 }
27
28 double j( void )
29 {
30     return acos(-1)*exp(1);
31 }
32
33 double f( int a )
34 {
35     return j() + a;
36 }
```

ERROR LISTING

```
oops.cxx:16:  error:  'j' was not declared in this scope
oops.cxx:21:  error:  cannot convert string to double
oops.cxx:23:  error:  void value not ignored as it ought to be
oops.cxx:33:  error:  new declaration 'double f(int)' ambiguates old declaration
undefined reference:  string ::g( double )
```

Beginning with question 21, different functions for you to write are described, each using arrays in a slightly different and increasing complex way. I have not attempted to provide space for the answers. Use an additional piece of paper, or better yet, a computer where you can test your solution's correctness. The solutions for all of these excercises are provided in one location at the end of the solution PDF. I will also provide a simple testing program for download (see **my** course website) so that you can easily test the correctness of your code.

21. In statistics, the mean or average value of a set of n data points is the defined by:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Write a C++ function, named `average`, that accepts two arguments: a `const` array of doubles, and a constant integer representing the number of elements in the double array. The function should return $\bar{x}$.

22. Write a void function that accepts six arguments: a `const int` array of data, an `const int size` argument for the array, an `int` argument named `value`, and two `int`s passed by reference named `near` and `far`. The function should place into the `near` and `far` the element offset of the array value *nearest* `value`, and the element offset of the value *farthest* from `value`.

For instance: if the argument is passed an array `{ 1, 13, 5, 8, 21, 2, 13, 3 }` with `value` 6, it should place into `near` the element offset of the element value 5, and into `far` the element offset of the element value 21.

23. The Fibonacci sequence is given by:

$$a_1 = a_2 = 1, \quad a_i = a_{i-1} + a_{i-2}$$

This sequence of numbers has a long history of study in both the mathematical[1] and biological world.[2] Now it's time to study them in the wonderful world of C++ arrays. Write a function, aptly named `fibonacci` that accepts an array of integers and a constant argument specifying the size of the array. `fibonacci` should initialize the array with as many terms of the Fibonacci sequence as it will hold.

24. Recall from Calculus II that all our favorite trigonometric functions have associated ∞-series (remember the names MacLaurin and Taylor?), for instance:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

(a) Write a void function named `sine_terms` that accepts a non-`const double` array (call it `terms`), a constant `size` argument for the array, and a `double` variable named `x`. The function should place into `terms` the first `size` terms of the MacLaurin series of $\sin x$.

(b) Calculus II also introduced the nth partial sum S_n of an ∞-series $\sum_{i=1}^{\infty} a_i$ as simply the sum of the first n terms:

$$S_n = \sum_{i=1}^{n} a_i$$

Now write the function `sine_sums` that accepts all the same arguments as in part a as well as a second non-`const double` array named `sums`. `sine_sums` should first call `sine_terms`, and then calculate the partial sums of the terms and place their values into `sums`. `sine_sums` should return the $S_{\texttt{size}}$ partial sum.

25. In mathematics, a *matrix* is a $p \times p$ grid of real ($\Re$) or complex ($\Re + \Im$) numbers. Each element of a matrix A is denoted by a_{ij} where i and j are the row and column *numbers*, respectively. The *transpose* of a matrix A is A^T, it is a matrix with A's elements reflected across the main upper-left to lower-right diagonal. These examples show the case for $p = 3$.

[1] It is connected to the golden ratio in an incredibly elegant way.

[2] The minimal packing pattern of sunflower seeds, the turns and ridges of sea shells, ...

$$\mathrm{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \mathrm{A}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

More succinctly, $A^T_{ji} = A_{ij}$. Write a C++ void function `transpose` that works on square matrices (the number of rows is equal to the number of columns) with a maximum dimension of `MAXP`. The matrix should be represented in C++ by a `MAXP`×`MAXP` double array. `transpose` should accept the array data, and a single constant integral parameter `p` ($1 \leq$`p`$\leq$`MAXP`) representing the size of the input array. `transpose` should turn the array argument into its transpose, this is called taking the transpose *in place*.

26. In mathematics, a vector $\boldsymbol{x}$ is a one dimensional array of numbers, typically visualized "vertically".
Vectors may multiply arrays (on the array's right-hand side) and the result is another vector. We write the following for the $p = 3$ case in mathematics:

$$\mathrm{A}\boldsymbol{x} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 \end{bmatrix}$$

Note that the product $\mathrm{A}\boldsymbol{x}$ is another vector of size p, each element of which is the sum of p products. The size, or dimension, of $\boldsymbol{x}$ must agree with the number of columns in A.[3]

Vectors in C++ are typically represented by 1d arrays. Complete the C++ void function below. `Ax` should calculate the product $\mathrm{A}\boldsymbol{x}$ for any arbitrary $1 \leq p \leq$`MAXP`.

```
void Ax( const double A[MAXP][MAXP], const double x[],
         double product[], const int p )
```

[3] For purposes of this worksheet, we are only interested in square matrices. So the dimension of $\boldsymbol{x}$ will also equal the number of rows in A. But this is not the general mathematical case! In general, a vector $\boldsymbol{x}$ of dimension n may multiply any matrix with dimensions $m \times n$, resulting in a new vector $\boldsymbol{y}$ that is m dimensional (the number of rows of A), and each component of $\boldsymbol{y}$ is the sum of n products.