Question 1 — 1 / 1 point

Given the function defined below, which line of code in the **main** function could be used to invoke it?

```
void heading (string name)
{       cout << "Acme Company\n";
        cout << "CEO: " << name;
}
```

✓ ○ heading ("Karen Loriz");

○ heading ( );

○ heading("Acme","Karen Loriz");

○ heading;

Question 2 — 0 / 1 point

Which variables are considered local variables to the function and will be stored in the function's activation record?

```
void heading (string name)
{       for (int num = 0; num < 5; num++)
              cout << name << endl;
}
```

○ name

✗ ○ num

⇨ ○ name and num

○ There are no local variables in this function.

Question 3 1 / 1 point

What is the output of the code?

```
void heading (int count);
int main ( )
{       heading (1);
        cout << "Hello\n";
        heading (2);
        return 0;
}
void heading (int count)
 {      for (int num=0; num<count; num++)
              cout << '*' << endl;
}
```

✓ ○
```
*
Hello
*
*
```

○
```
*
*
*
Hello
```

○
```
*
Hello
*
```

○
```
*
Hello
*
*
Hello
```

Question 4 1 / 1 point

The code below contains an error since the argument and the parameter have the same names.

```
void display (int count);
int main ( )
{       int count = 2;
        display(count);
        return 0;
}
void display (int count)
 {      for (int num=0; num<count; num++)
              cout << "Hello\n";
}
```

○ True

✓ ○ False

Question 5 1 / 1 point

The scope of a **global constant** extends to all functions in a source code file (ex. Source.cpp).

✓ ○ True

○ False

Question 6 0 / 1 point

What are the best comments for the **division** function?

```
double division (double num, double den)
{       return num / den;
}
```

○ // PRE: num != den
// POST: return the quotient of num divided by den

○ // PRE: pass in two doubles
// POST: return the quotient of num divided by den

✗ ○ // PRE: none
// POST: return the quotient of num divided by den

➡ ○ // PRE: den != 0
// POST: return the quotient of num divided by den

Question 7 1 / 1 point

Which is the most appropriate way to invoke the **mystery** function?

```
void mystery (int a, int& b);
int main ( )
```

```
{      int k = 5;
        _____
        return 0;
}
```

- [✓ ◯] mystery (4, k);
- ◯ mystery (5, "Hello");
- ◯ cout << mystery (5, 4);
- ◯ mystery (5, 4);

Question 8 1 / 1 point

Which of the following should be the function prototype for a function **mystery** which will use two reference parameters to integers? The function does not return any value.

- ◯ int& mystery (int num1, int num2);
- ◯ void mystery (int num1, int num2);
- ◯ int mystery (int& num1, int& num2);
- [✓ ◯] void mystery (int& num1, int& num2);

Question 9 1 / 1 point

What is the output of the code?

```
void f (int& m, int p);
int main ( )
{   int w = 3, z = 5;
    f (w, z);
    cout <<  w << " " << z ;
    return 0;
}
void f  (int& m,  int p)
{   int g = 2;
    p++;
    m = g * p;
}
```

- 3 6
- 3 5
- [✓ 12 5]
- 12 6

Question 10 1 / 1 point

Which statement best describes identifier **m**?

```
void f (int& m, int p);
int main ()
{   int w = 3, z = 5;
    f (w, z);
    cout <<  w << " " << z ;
    return 0;
}
void f  (int& m,  int p)
{   int g = 2;
    m = g * p;
}
```

- [✓ **m** is a reference created when function **f** is called associated with variable **w**]
- **m** is a reference created when function **f** is called associated with variable **p**
- **m** is an integer variable created when function **f** is called and destroyed when function **f** ends
- **m** is an integer variable that has a lifetime of the entire program

Question 11 1 / 1 point

What is the output of the code?

```
void display (string n, string j = "Sales");
int main ( )
{       display ("Tia");
        display ("Sal", "Clerical");
        return 0;
}
void display (string n, string j)
{       cout << n << " " << j << endl;
}
```

- ○ Tia
  Sal Clerical
- ✓ ○ Tia Sales
  Sal Clerical
- ○ Tia Sales
  Sal Sales
- ○ This code generates an error

Question 12 0 / 1 point

Which statement is not true about use of reference parameters in software development?

- ⇨ ○ Reference parameters should be used whenever possible as they speed up execution time
- ✗ ○ Typically, programmers use value parameters more often than reference parameters
- ○ Reference parameters should be used when the function body must make changes to the argument variable
- ○ Use of reference parameters can make debugging more challenging

Question 13 1 / 1 point

A program uses the function prototype seen below. Which second function prototype, if added to the program, would be considered an overloaded function?

double f (int n1, int n2);

- ◯ double func (int n1, int n2);
- [✓ ◯ double f (double n1, double n2);]
- ◯ int f (int n1, int n2);
- ◯ void f2 ( );

Question 14 1 / 1 point

What is the output of the code?

```
void swap (int & v1, int & v2);
int main ( )
{      int n1 = 10, n2 = 20, n3 = 30;;
        swap (n1, n2);
        swap (n2, n3);
        cout << n1 << " "  << n2 << " " << n3;
    return 0;
}
void swap (int & v1, int & v2)
{      int temp = v1;
        v1 = v2;
        v2 = temp;
}
```

- [✓ ◯ 20 30 10]
- ◯ 30 10 20
- ◯ 10 20 30

- ○ 20 10 30

Question 15 1 / 1 point

Which of the following *is not* a valid enumeration?

- ○ enum class Pet {DOG = 0, CAT = 1, RABBIT = 2, BIRD = 3};
- ○ enum class Status {WIN,LOSE,CONTINUE};
- ○ enum class Person {ME, YOU};
- ✓ ○ enum class Day {"MON", "TUE", "WED"};

Question 16 1 / 1 point

What happens when two blocks, one nested inside of the other, both declare variables with the same identifier? Assume that the outer block declares its variable before the opening left-brace of the inner block.

- ○ The "outer" variable is irretrievably lost when the "inner" variable is declared
- ✓ ○ The "outer" variable is hidden while the "inner" variable is in scope
- ○ A syntax error occurs
- ○ The "inner" declaration is ignored and the "outer" variable has scope even inside the inner block

Question 17 1 / 1 point

An activation record for function f will be *popped off* the function call stack when _____.

- ✓ ○ function **f** returns control to its caller
- ○ function **f** begins execution
- ○ function **f** calls another function
- ○ the main function completes and the program ends

Question 18 1 / 1 point

A program has a global constant named ERR and a local constant also named ERR in function **f**. How could global ERR be accessed with function f?

- ○ ERR
- ✓ ○ ::ERR
- ○ &ERR
- ○ The global constant cannot be accessed due to the name conflict

Question 19 1 / 1 point

Overloaded functions *must* have ____.

- ○ different return types
- ✓ ○ different parameter lists
- ○ the same number of parameters
- ○ different function names

Question 20 0 / 1 point

Given the following function template what would be returned by the two function calls?

```
maximum(2, 5);
maximum(2.3, 5.2);

template <typename T>
T maximum(T value1, T value2)
{  if (value1 > value2)
      return value1;
   else
      return value2;
}
```

- ○ 5 and a type-mismatch error
- ➡ ○ 5 and 5.2
- ○ 2 and 2.3
- ✗ ○ two error messages