True/False (1pt each)

1. (**True**/False) Overloading a binary operator as a member only requires one argument.

2. (True/**False**) Global variables are a good programming practice.

3. (**True**/False) You should implement your code incrementally, testing as you go.

4. (**True**/False) When passing objects by reference it is a good idea to make the object a `const`.

5. (True/**False**) In C++ you must declare a default constructor in a class or the class will not be able to be instantiated.

6. (True/**False**) Line (3) produces a memory leak of 1 byte because `cprt` is never deleted.

```
(1)     char c = '1';
(2)     char *cprt;
(3)     cprt = &c;
```

7. (True/**False**) In C++ the compiler will infer the type intended for a variable from the context in which the variable occurs.

8. (True/**False**) You should wait until the day before an assignment is due to actually start coding it.

9. (**True**/False) Line (2) is a legal initialization of the `dp` pointer.

```
(1)     double *d;
(2)     double *dp = &d;
```

10. (**True**/False) There is no problem with these two function definitions (the C++ compiler will be able to distinguish between the two):

```
int sum(double x, int y){/*…*/ return something_int;}
int sum(int x, double y){/*…*/ return something_int;}
```

11. (True/**False**) Child classes (classes that inherit from some parent class) have direct access to all of the parent's private member variables.

12. (True/**False**) Child classes (classes that inherit from some parent class) inherit all of their parent's constructors.

13. (True/**False**) When redefining a parent's method in the child class you must change the function signature or the C++ compiler will not know which method to use.

14. (True/**False**) Overloading a binary operator as a member requires two arguments.

15. (True/**False**) Nonconstant objects can only call nonconstant methods.

16. (True/**False**) A friend function has access to that class's public member variables/methods but not their private member variables/methods.

17. (**True**/False) As a programmer, when you are testing individual methods you should make sure to test both valid inputs as well as invalid inputs.

18. (True/**False**) Singly linked lists do not have random access but doubly linked lists do.

19. (True/**False**) The following segment of code produces a memory leak because `cprt` is never deleted.

```
(1)     char c = '1';
(2)     char *cprt;
(3)     cprt = &c;
```

Multiple Choice

1. In which of the following is y not equal to 5 after execution? Assume x is equal to 4.
   a. y = 5;
   b. y = ++x;
   **c. y = x++;**
   d. y = x = 5;

2. When an argument is passed call-by-value, changes in the calling function _____ affect the original variable s value; when an argument is passed call-by-reference, changes _____ affect the original variable s value, in the calling function.
   a. do, do not
   b. do not, do not
   c. do, do
   **d. do not, do**

3. Referencing elements outside the array bounds
   **a. can result in changes to the value of an unrelated variable**
   b. will always cause a core dump
   c. will cause the code not to compile
   d. correctly enlarges the size of the array

4. What will the following program segment do if values of 2.3 and 3.2 are entered at the stdin?

```
int i, j;
i = j = 1;

cin >> i >> j;
cout << i << " " << j << endl;
```

   a. 1 1
   **b. 2 0**
   c. 2.3 3.2
   d. Will not print anything due to a runtime error.

5. Given the following function template:

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

   What would be returned by the following two function calls?

```
maximum('c', 'f'); maximum(2.3, 5.2)
```

   **a. 'f' and 5.2**
   b. No output, 5.2
   c. 'f', No output
   d. No output, No output

6. Consider the following code

```
int x = 12;
if( 8 && (x = 6) )
      cout << x;
```

   What is the output?

   **a. 12**
   b. 6
   c. Nothing (if statement is not executed)
   d. Syntax error (will not compile and run)

7. What are the outputs of the following code:

```
int x(0), y(0);
for (x = 2; x < 3; x++)
      cout << (y=x) << endl;
cout << x << endl;
```

   a. True, 3
   b. False, 2
   **c. 2, 3**
   d. No outputs

8. What are the outputs of the following code:

```
int i = 55;
int *iPtr;
iPtr = &i;
*iPtr = 92;
cout << i << endl;
cout << *iPtr << endl;
```

   a. 55, 92
   b. 55, 55
   **c. 92, 92**
   d. Error, will not compile or will be a memory leak

9. What will be the output of the following piece of code?

```
int a = 7, b = 9, c = 10;
if (a > b)
      a = 4;
      if ( c > b)
            a = 5;
else
      a = 6;
cout << a < endl;
```

   **a. 5**
   b. 4
   c. 7
   d. 6

10. Which of the following overloadings of the + operator will be invoked by the call in line (2)?

```
(1) String s();
(2) s + "hello";
```

   a. `String & operator+(String &);`

   b. `String & operator+(int);`

   **c. `String & operator+(string);`**

   d. `String & operator+(String &, string);`

11. Consider the following statement:

```
int *temp = new int[10];
delete [] temp;
```

The `delete` keyword DOES NOT

**a) Set the values of `temp` to NULL**

b) Deallocate memory where `temp` is pointing

c) Need to be called in this case (without it there would be no memory leak)

d) Use proper syntax here (it should read: `delete temp [10]`)

12. Assume you have a myArray object that is templated with the generic class T. Which of the following is the proper overloading for line (2)?

```
(1) myArray<int> intArray1, intArray2;
(2) intArray2 = intArray1 + 4;
```

a) `myArray operator+(T*);`

**b) `myArray operator+(T);`**

c) `myArray operator+(T[]);`

d) `void operator+(T);`

13. `y` and `z` are user-defined objects and the += operator is an overloaded member function. The operator is overloaded such that `y += z` adds `z` and `y`, then stores the result in `y`. Which of the following expressions is equivalent to `y += z`?

a) `y =(y.operator+=)+(z.operator+=)`

**b) `y.operator+=(z)`**

c) `y = y.operator+=(z.operator+=())`

d) `y.operator+=(z)=y.operator+=(z)+z.operator+=(z)`

14. In the following segment of code how many bytes of memory are lost (assuming an int occupies 4 bytes)?

```
int i(100), *iPtr;
iPtr = &i;
```

a) 4 bytes

b) 8 bytes

c) 12 bytes

**d) 0 bytes**

15. Which of the following operator overloadings are cascade capable? (Circle all correct answers)

a) **`myArray & operator=(myArray);`**

b) `double* operator+(myArray);`

c) `double operator+(myArray);`

d) `void operator=(myArray);`

16. Which of the following IS NOT dynamic data structures?

**a) primitive C++ arrays**

b) linked list

c) your myArray object

d) heap

Free Answer - the final will be multiple choice. We will discuss how these questions below would turn into MC on 12/01 at the review session.

17. (2pts) Consider the following header:

```
class IntPair
{
	private:
		int first;
		int second;
	public:
		IntPair(int firstValue, int secondValue);
		const IntPair operator++( ); //Prefix version
		IntPair operator++(int); //Postfix version
		int getFirst( ) const;
		int getSecond( ) const;
};
```

Is the following legal? Why or why not?

```
IntPair a(1,2);
//(a++)++;
```

++(a++);  //illegal - prefix returns a const object that cannot call a non-const function (postfix)

18. (3pts) Fill out the memory diagram with the final values of each variable. Is there any memory leaked? If so, how many bytes? If not, explain why there is no leak.

```
double d(0.0), *dptr;
dptr = &d;
*dptr = 20;
```

| Name | Address | Content |
|---|---|---|
| d | 0x100 | 20 |
| dptr | 0x104 | 0x100 |

19. (5pts) Look over the following header and answer the questions that follow.

```
const myArray& operator= (const myArray& rhs);
```

a. (2pts) Give two reasons why the argument to the operator= is passed by reference.
-Memory efficiency - do not have to create an entire temporary object It is safe - the object passed in will always be in existence for the scope of the call

b. (1pts) Explain why the argument to the operator= is passed as const.
-When the = operator is called we do not want the argument (right hand side of the equation) to be modified.

c. (1pts) Explain why the operator= has a return type, the same as its argument.

- We return the value object to set the myArray object equal to it and it can be cascade capable

d. (1pts) Explain why the returned object is a const.

- When the value is returned it does not need to be changed.

20. (5pts) List three different types of constructors and describe the identifying attribute for each.

- Parameterized constructor takes in values for parts of the object, for example the size and a data vale
- Copy constructor copies an already created object so they are the same
- A default constructor takes in no parameters and creates the most basic form of this object.

21. (8pts) Consider the following node object:

```
class node {

public:
    node();
    node(int);
    node(int, node *);
    node(const node&);
    ~node();

    void printAll();
    node* GetNextNode() const;
    void SetNextNode(node*);
    int GetValue() const;
    void SetValue(int );

private:
    int value;
    node *nextNode;

};
```

Write code to connect ten nodes together to form a singly linked list. Then write code to insert a node at index 2.

```
node * head, *temp, *curr;
 head = new node;
curr = new node;
head->nextNode = curr;
for(int i = 1; i < 10; i++){
      temp = new node;
      curr->nextNode = temp;
      curr = temp;
}

//inserting ->
curr = head->nextNode;
temp = new node;
temp->nextNode = curr->nextNode;
curr->nextNode = temp;
```

22. (2pts) Write a loop (your choice) that prints all numbers divisible by 2 in the range 0 to 10 (inclusive). You can print to stdout using cout.

```
for(int i=0; I<= 10; i++)
{
          if(i%2 == 0)
                    cout << i;
}
```

23. (2pts) What is the meaning of “cascading” in the context of overloading operators in C++? Why is it useful?

Cascading means that multiple functions can be used in the same line. This is useful as you can do an object plus another object plus another object all in the same line and have that be able to set to an object. This makes code more efficient and multiple functions to take place on the same line.

24. (4pts) Write a function that takes in three ints and sends the max and min of those three numbers back to the main to be stored in `max` and `min`. Write the prototype and implementation. Assume the following main.

```
//declare your function prototype here

Void findMaxMin(int,int,int,int,int);


int main(int argc, char **argv)
{
      int a(10), b(-1), c(6), max(-999), min(999);
      findMaxMin(a,b,c,max,min);
      cout << "max: " << max << endl;
      cout << "min: " << min << endl;
}
```

```
      //implement your function here



Void findMaxMin(int a,int b, int c, int max, int min)
{
      if ( a < b && b < c)
      {
            min = a
            max = b
      }
      Else if ( a < b && c < b)
      {
            min = a
            max = c
      }
      Else if ( b < a && a < c)
      {
            min = b
            max = c
      }
      Else if ( b < a && c < a)
      {
            min = b
            max = a
      }
      Else if ( c < a && a < b)
      {
            min = c
            max = b
      }
      Else if ( c < a && b < a)
      {
            min = c
            max = a
      }
}
```

25. (3pts) The following piece of code has memory leak(s). Identify where the memory leak(s) is/are, how many bytes were lost and indicate how you would fix it/them.

```
int main(int argc, char **argv) {
    char *data;
    int *a, b(10);
    a = new int; //memory leak -> never deleted. 4 bytes
    a = &b;
    data = new char[5];
    delete [] data; //memory leak -> should have a [] 4 bytes
    delete data;
    return 0;
}
```

26. (2pts) What is the difference between a private and public function? Why might you want to use a private function?

A public function is accessible anywhere within the code, however something private is only accessible through an object. This means that a private object is more restrictive on when it can be called in the code.

27. (3pts) Suppose you have the following array declaration in a program.

```
int yourArray[10];
```

Further suppose that in the implementation of C++ you are using an int that requires 4 bytes.

a) When your program runs, how much memory is required for this array? **40 bytes**

b) Suppose further that your array starts at memory location decimal 100. What will be the address of `yourArray[5]`? **120**

c) If you wrote to the (illegal) index 11 position in `yourArray` to what address would this clobber? **144**