NAME: _____

```
int i = 27;
while (i >= 3) {
      i = i / 3;
}
cout << i * 2 - 3;
```

Figure 1

1) When the code in Figure 1 runs, what does it output to the console? (25 points)

```
int n = 3;
int k = n++ * 2;
int j = ++n * 2 + k;
cout << j / n;
```

Figure 2

2) When the code in Figure 2 runs, what does it output to the console? (25 points)

3) Write code that prints 500 random integers that are each less than 1000. (25 points)

4) Suppose that you are in a console window on a Linux computer and your current directory is your home directory. In the spaces provided below, show the command to perform the desired operations. (25 points)

a. Create a folder named *sam* in your home directory.

b. Suppose that a file named *main.cpp* is located in your home directory. Copy this file to the *sam* directory that you created in the previous problem.

c. List the contents of the *sam* directory.

d. Rename the file *main.cpp* in your home directory to *ex1.cpp*.

```
int k = 99;
for (int i = 1; i < k + 1; ++i)
{
     cout << "hi again";
}
```

Figure 3

5) How many times does the code in Figure 3 print "hi again"? (25 points)

6) Write code that computes the sum of integers 99 through n, inclusive, where n is an integer strictly greater than 99. (25 points)

```
bool areEqual(int a[ROWS][COLS], b[ROWS][COLS]);
```

Figure 4

7) Implement the function *areEqual* whose declaration appears in Figure 4. Both arguments of the function are 2-dimensional arrays of integers. The function returns true if each value in *a* is equal to the value at the same row/column position in *b*. The variables ROWS and COLS are constants defined elsewhere in the program; you don't need to define them, just use them. (25 points)

```
int i = 7;
for (int k = 0; k < 300; ++k)
{
      i = i + 3;
}
cout << i;
```

Figure 5

8) When the code in Figure 5 runs, what does it output to the console? (25 points)

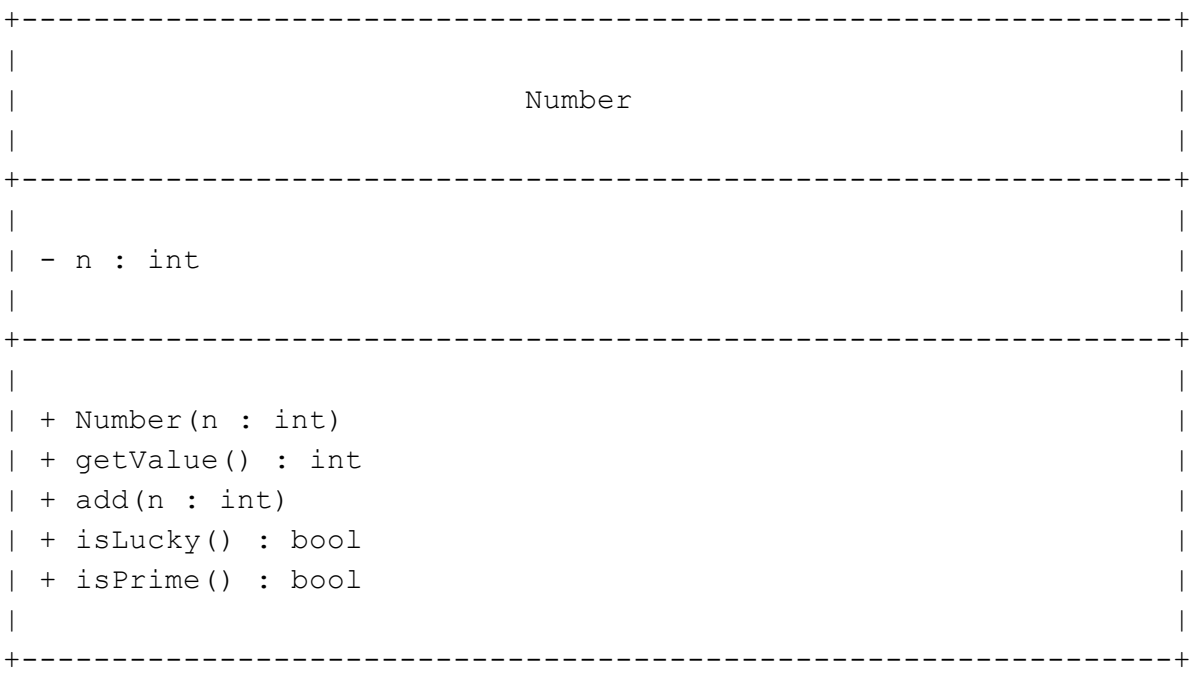

Figure 6

Instances of the *Number* class represent integers. The integer value they represent is passed into the constructor and stored in member variable *n*. The *getValue* function returns the number the object currently represents. The function *add* modifies the state of the number by adding the value passed into the function. The *isLucky* function returns true if the value is a multiple of 7 and not a multiple of 13. For example, the number 2 * 7 * 13 = 182 is not lucky because 13 is a factor. The *isPrime* function returns true if the value is a prime number, otherwise it returns false.

9) Provide an implementation of the constructor for the Number class. (25 points)

10) Provide an implementation of the *getValue* function in the *Number* class. (25 points)

11) Provide an implementation of the *add* function for the *Number* class. (25 points)

12) Provide an implementation of the *isLucky* function for the *Number* class. (25 points)

13) Provide test code for the *isLucky* function. Use *assert* statements for this purpose. Make sure that your test code executes every line of code in the function. (25 points)

14) Provide an implementation of the *isPrime* function for the *Number* class. (25 points)

```
void cleanse(vector<int> & v);
```

Figure 7

15) Implement a function that replaces in a vector each occurrence of the number 0 with the number 1 and the number 13 with the number 7 and leaves all other values unchanged. For example, the function would convert (13, 1, 0, 26, 7) to (7, 1, 1, 26, 7). A declaration of the function is shown in Figure 7. (25 points)

```
int minValue(const vector<int> & v);
```

Figure 8

16) Provide an implementation of the minValue function whose declaration is shown in Figure 8. The function takes a single argument, which is a vector of int. The function returns the smallest int that is in the vector. (25 points)

```
int search(const vector<int> & v, int k);
```

Figure 9

17) Implement a function that searches for a given value in a vector of integers. If the value is found, the function returns the index of the value in the vector; otherwise it returns -1. Do not assume the values are in order; do not use binary search. For example, for v = (-2, 4, 18, 6) the function returns -1 for k = 1 and 2 for k = 18. A declaration of the function is shown in Figure 9. (25 points)

```
bool binarySearch(int a[], int len, int k);
```

Figure 10

18) Implement a function that uses binary search to search for a given value *k* in an array of integers whose elements are in strictly increasing order. If the value is found, the function returns true, otherwise it returns false. A declaration of the function is shown in Figure 10. The second argument, *len*, is the number of integers in the given array.
(25 points)

```
int countOccurrences(const vector<int> & v, int k);
```

Figure 11

19) Implement the function *countOccurrences* whose declaration appears in Figure 11. The first argument of the function is a vector *v* of integers and the second argument is an integer *k*. The function returns the number of times *k* occurs in *v*. (25 points)

```
void mysort(vector<int> & v);
```

Figure 12

20) Write a function called *mysort* that rearranges elements of a vector *v* so that they form a strictly increasing sequence of values. Do not use a predefined sort routine from the standard library. A declaration of the function is shown in Figure 12. (25 points)