NAME: ____________________________________

```
int i = 64;
while (i >= 2) {
     i = i / 2;
}
cout << i * 3 - 1;
```

Figure 1

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

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

Figure 2

2) Write a function named *makeLucky* that takes a reference to a vector of int. The function replaces each element that is a multiple of 13 and not a multiple of 7 into a zero. For example, the function transforms vector (91, 14, 4, 26) into (91, 14, 4, 0). Because 91 is a multiple of 7 and 13, we leave it alone; however, 26 is a multiple of 13 and not of 7 so we replace it with a zero. The numbers 14 and 4 are neither multiples of 13, so we leave them alone. The declaration of the function is given in Figure 2. (25 points)

```
int k = 123;
for (int i = 1; i <= k + 1; ++i)
{
     cout << "hi again";
}
```

Figure 3

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

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

Figure 4

4) 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 = 4;
for (int k = 0; k < 900; ++k)
{
     i = i + 3;
}
cout << i;
```

Figure 5

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

```
+----------------------------------------------------------------+
|                                                                |
|                            Number                              |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
| - n : int                                                      |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
| + Number(n : int)                                              |
| + getValue() : int                                             |
| + add(n : int)                                                 |
| + isLucky() : bool                                             |
|                                                                |
+----------------------------------------------------------------+
```

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.

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

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

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

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

10) 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)

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

Figure 7

11) 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)

12) Call multiples of 13 unlucky numbers. Write code that computes the sum of unlucky numbers between *0* through *9000*. (25 points)

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

Figure 8

13) 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 8. (25 points)

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

Figure 9

14) 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 the index of the value in the array, otherwise it returns -1. A declaration of the function is shown in Figure 9. The second argument, *len*, is the number of integers in the given array. (25 points)

15) Write a program that implements Guess-the-Number game. The program should enter a loop that starts by printing “What is the number?” After printing this, it reads the user response. (Use *cin* >> *n* to read the user response.) If the user enters a value less than 1023, the program prints “too small” and continues the loop. If the user enters a number larger than 1023, the program prints “too big” and continues the loop. If the user enters the number 1023, the program prints “you got it” and then terminates. (25 points)

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

Figure 10

16) Write a function called *mysort* that rearranges elements of a vector *v* so that they form a strictly **decreasing** sequence of values. Do not use a predefined sort routine from the standard library. A declaration of the function is shown in Figure 9. (25 points)