Name: _____

Write all of your responses on these exam pages. If you need extra space please use the backs of the pages.

1 Short Answer

1. (*5 Points*) Write a function named `poparr` that takes in a single integer parameter named `size`. The function will create a dynamic array of floats, populate the array with random numbers between 0 and 1, and finally return a pointer to the array.

2. (*5 Points*) Write a segment of code that will. Create a dynamic array of integers of size 1000, call is A. Populate the array with random numbers between 1 and 100. Print out the array to the screen in one line of output. Reverse the order of the array without using a second array, that is, just move the elements of A around. Then shift all of the elements of the array one entry to the left putting a 0 in the right most entry. Finally clear the array from memory. **All of this is to be done using pointer arithmetic for array access, (no A[i]).**

3. (*10 Points*) Given the following class. In the specification, add in the prototypes for overloading + and *. The + will return a vec3 type and the * will return a double. Then implement these two functions outside the specification.

   The + will add the x values of the two objects, the y values of the two objects, and the z values of the two objects. The * will multiply the x values, multiply the y values, and multiply the z values, and then add them all up. So if $a = (1, 2, 3)$ and $b = (4, 5, 6)$ then $a+b = (5, 7, 9)$ and $a*b = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$. So the code on the left will produce the output on the right.

```
vec3 a(1, 2, 3);
vec3 b(4, 5, 6);
vec3 c = a + b;
a.print();
b.print();
c.print();
float d = a * b;
cout << d << endl;
```

```
1 2 3
4 5 6
5 7 9
32
```

```
class vec3 {
private:
    double x, y, z;

public:
    vec3(double xv = 0, double yv = 0, double zv = 0) {
        x = xv; y = yv; z = zv; }

    void print() { cout << x << " " << y << " " << z << endl; }

    // Prototypes for overloaded + and *




};
```

4. (*10 Points*) Answer the following questions.

   (a) What math operations are allowed on pointers?

   (b) Assuming that ptr is a pointer to an int, what happens when you add 4 to ptr?

   (c) Under what circumstances can you successfully return a pointer from a function?

   (d) What is a mutator function? What is an accessor function?

   (e) What is a default constructor? Is it possible to have more than one default constructor?

   (f) Is it possible to have more than one constructor? Is it possible to have more than one destructor?

   (g) Describe the difference between an instance member variable and a static member variable.

   (h) When is a copy constructor called?

   (i) What is passed to the parameter of a class's operator= function?

   (j) Why shouldn't a class's overloaded = operator be implemented with a void operator function?

2 Program Trace

1. (*10 Points*) Write the output of the following program given the inputs specified.

```
#include <iostream>
using namespace std;

int* fnc1(const int*, int);
int* fnc2(int*, int);
void displayArray(const int*, int);

int main() {
    const long SIZE = 7;
    int *A = new int[SIZE];
    int *a, *b, *B;

    cout << "Input: ";
    for (int i = 0; i < SIZE; i++)
        cin >> A[i];

    int n = *(A + 2);
    if (n > 0)
        B = fnc1(A, SIZE);
    else
        B = fnc2(A, SIZE);

    displayArray(A, SIZE);
    displayArray(B, SIZE);

    a = B;
    cout << *a << endl;
    a += 4;
    cout << *a << endl;
    cout << ++(*a) << endl;
    a++;

    b = A + 1;
    cout << *b << endl;
    cout << *b++ << endl;
    cout << *b << endl;

    delete[] A;
    return 0;
}

int* fnc1(const int *a, int size) {
    int *newArray = new int[size];
    for (int i = 0; i < size; i++)
        newArray[i] = *(a + (i + 1) % size);

    return newArray;
}

int* fnc2(int *a, int size) {
    int *p = a;
    while (p < &a[size]){
        if (*p % 2 == 0)
            *p = *p / 2;
        else
            *p = *p * 3 + 1;
        p++;
    }

    return a;
}

void displayArray(const int arr[], int size) {
    for (int index = 0; index < size; index++)
        cout << arr[index] << " ";
    cout << endl;
}
```

(a) Input: -1 3 -2 4 -3 5 10

(b) Input: 3 5 7 9 11 13 15

(c) Is there a memory leak in this program? If so, where and what circumstances do we have a leak?

2. (*10 Points*) Write the output of the following program.

```
#include <iostream>
using namespace std;

class Thing1 {
private:
    int a, b;

public:
    Thing1(int x = 1, int y = 2) {
        a = x; b = y;
    }

    void set(int x = 1, int y = 2) {
        a = x; b = y;
    }

    int geta() {
        return a;
    }

    int getb() {
        return b;
    }

    void print() {
        cout << a << " " << b << endl;
    }
};

class Thing2 {
private:
    int a, b, c;
    Thing1 *A;

public:
    Thing2(int x = 1, int y = 2, int z = 3) {
        a = x; b = y; c = z;
        A = new Thing1[c];
        for (int i = 0; i < c; i++)
            A[i].set(i % a, i % b);
    }

    int geta() {
        return a;
    }

    int getb() {
        return b;
    }

    int getc() {
        return c;
    }

    int geta(int i) {
        return A[i].geta();
    }

    int getb(int i) {
        return A[i].getb();
    }

    void print() {
        cout << a << " " << b << " " << c << endl;
        for (int i = 0; i < c; i++)
            A[i].print();
    }

};
```

```
int main() {
    Thing1 t1;
    Thing2 t2(4, 5, 6);
    Thing1 t3(7, 11);

    t1.print();
    t2.print();
    t3.print();

    cout << t1.geta() << endl;
    cout << t2.geta() << endl;
    cout << t3.getb() << endl;
    cout << t2.getb() << endl;
    cout << t2.getb(4) << endl;
    cout << t2.getb(5) << endl;
    cout << t2.getb(t1.geta()) << endl;

    return 0;
}
```

Output:

Is there a memory leak in this program? If so, where and what circumstances do we have a leak?

3 Coding

This exercise is to write a complete class structure named Simulator. This will be using dynamic memory allocation so make sure that you code does not produce any memory leaks.

The data of the class is below with descriptions. All data is to be private, as usual.

- maxrandom — Integer storing the maximum random number used in the simulation.
- simlength — Integer storing the length of the simulations, i.e. the number of trials.
- simulation — Pointer to an array holding the entire simulation. That is, an array of length simlength that will hold random numbers from 1 to maxrandom.
- counts — Pointer to an array holding the counts of the occurrences of values in the simulation. That is, an array of length maxrandom that will hold the counts for each of the random numbers from 1 to maxrandom.

The class will also have the following functions and what they do. RunSimulation and Clear are to be private and the rest are to be public. All functions are implemented outside the specification, as if they were contained in the Simulator.cpp file.

- Constructor — Have at least two constructors, one that is a default that sets the simulation run to 100 and the maximum random number to 6. Also have another constructor take the run length and maximum random number in as parameters. The constructors should call RunSimulation so that the arrays are always populated with the last simulation run.
- Destructor — Should clear all allocated memory so that there are no memory leaks.
- Copy Constructor — A standard copy constructor for the data items being stored.
- Overloaded Assignment (=) — A standard overloaded assignment operator for the data items being stored.
- Overloaded stream out (<<) — This will print out the simulation on one line and counts on another. For example,

```
Simulation: 2 3 4 2 2 2 3 1 4 2 3 4 3 2 1 1 1 3 3 1
Counts: 5 6 6 3
```

- reset — This will reset the simulation. That is, reset all the data and rerun the simulation.
- Stats — This function will print out the statistics from the last run of the simulation, reporting the length, the random number range, the maximum counts and the minimum counts as well as the numbers that have the maximum and minimum counts. The stats output from the example above is,

```
Simulation Length: 20
Simulation Range: 1 to 4
Maximum count is: 6 which occurs at value(s) 2 3
Minimum count is: 3 which occurs at value(s) 4
```

- RunSimulation — This will run the simulation using the current length and maximum. It will populate the simulation array with random numbers between 1 and the maxrandom value, inclusive. It will then create an array of counts for each number between 1 and the maxrandom value.
- Clear — This will populate both simulation and counts with 0.

A sample main and its output are below.

Sample Main

```
int main() {
    Simulator sim(20, 4);
    cout << sim << endl;
    sim.Stats();
    cout << "------------" << endl;
    Simulator sim2 = sim;
    cout << sim2 << endl;
    sim2.Stats();
    cout << "------------" << endl;
    sim2.reset(10, 2);
    cout << sim << endl;
    cout << sim2 << endl;
    cout << "------------" << endl;
    sim = sim2;
    cout << sim << endl;
    cout << sim2 << endl;
    cout << "------------" << endl;
    sim.reset(15, 6);
    cout << sim << endl;
    cout << sim2 << endl;
    return 0;
}
```

Output

```
Simulation: 2 3 4 2 2 2 3 1 4 2 3 4 3 2 1 1 1 3 3 1
Counts: 5 6 6 3

Simulation Length: 20
Simulation Range: 1 to 4
Maximum count is: 6 which occurs at value(s) 2 3
Minimum count is: 3 which occurs at value(s) 4
------------
Simulation: 2 3 4 2 2 2 3 1 4 2 3 4 3 2 1 1 1 3 3 1
Counts: 5 6 6 3

Simulation Length: 20
Simulation Range: 1 to 4
Maximum count is: 6 which occurs at value(s) 2 3
Minimum count is: 3 which occurs at value(s) 4
------------
Simulation: 2 3 4 2 2 2 3 1 4 2 3 4 3 2 1 1 1 3 3 1
Counts: 5 6 6 3

Simulation: 2 2 1 2 1 2 2 1 2 1
Counts: 4 6

------------
Simulation: 2 2 1 2 1 2 2 1 2 1
Counts: 4 6

Simulation: 2 2 1 2 1 2 2 1 2 1
Counts: 4 6

------------
Simulation: 4 1 1 3 3 5 6 1 5 6 4 4 5 6 5
Counts: 3 0 2 3 4 3

Simulation: 2 2 1 2 1 2 2 1 2 1
Counts: 4 6
```

1. (*5 Points*) Write the specification for the Simulator class. That is, what you would put in the Simulator.h file, you do not need to guard it here. No inline code at all.

2. (*5 Points*) Write the constructor(s).

3. (*5 Points*) Write the destructor.

4. (*5 Points*) Write the copy constructor.

5. (*5 Points*) Write the overloaded = operator, assignment.

6. (*5 Points*) Write the RunSimulation function.

7. (*5 Points*) Write the Clear function.

8. (*5 Points*) Write the stream out (<<) operator.

9. (*5 Points*) Write the reset function.

10. (*5 Points*) Write the Stats function.