

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.

Solution:

```
float* poparr(int size) {
    float *arr = new float[size];
    for (int i = 0; i < size; i++)
        arr[i] = 1.0 * rand() / RAND_MAX;
    return arr;
}
```

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]).**

Solution:

```
int size = 1000;
int *A = new int[size];

for (int i = 0; i < size; i++)
    *(A + i) = rand() % 100 + 1;

for (int i = 0; i < size; i++)
    cout << *(A + i) << " ";

for (int i = 0; i < size / 2; i++) {
    int temp = *(A + i);
    *(A + i) = *(A + size - i - 1);
    *(A + size - i - 1) = temp;
}

for (int i = 0; i < size - 1; i++)
    *(A + i) = *(A + i + 1);
*(A + size - 1) = 0;

delete[] A;
```

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

```

Solution:

```

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; }

    double operator *(const vec3&);
    vec3 operator +(const vec3&);
};

vec3 vec3::operator +(const vec3 &right) {
    vec3 temp;

    temp.x = x + right.x;
    temp.y = y + right.y;
    temp.z = z + right.z;

    return temp;
}

double vec3::operator *(const vec3 &right) {
    double retval = x * right.x + y * right.y + z * right.z;
    return retval;
}

```

4. (10 Points) Answer the following questions.

(a) What math operations are allowed on pointers?

Solution: +, -, +=, -=, ++, --.

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

Solution: ptr now points to the position in memory that is 4 integers further. That is 4*sizeof(int) bytes.

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

Solution: If the pointer/memory was created in the function. Also if a pointer variable is passed as a parameter it can be returned as well.

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

Solution: Mutators change the class data, member data (sets). Accessor functions just return member data and do not alter it (gets).

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

Solution: The default constructor is one that takes no parameters. There can be only one.

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

Solution: Yes there can be more than one constructor but there is only one destructor.

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

Solution: An instance member variable is different for all instances of a class, that is, are stored in different locations in memory. A static member variable is shared between all instances of the class.

- (h) When is a copy constructor called?

Solution: When construction is done and the new object is initialized with another object. In addition, when a function parameter is a class structure and is sent by value.

- When an object of the class is returned by value: `return obj;`
- When an object of the class is passed to a function by value as an argument: `fct(MyObj obj).`
- When an object is constructed based on another object of the same class: `MyObj obj = obj2.`
- When compiler generates a temporary object.

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

Solution: The right hand object. That is, `A = B`, `B` would be the parameter value.

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

Solution: You could not do `A = B = C`.

2 Program Trace

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

```

1  #include <iostream>
2  using namespace std;
3
4  int* fnc1(const int*, int);
5  int* fnc2(int*, int);
6  void displayArray(const int*, int);
7
8  int main() {
9      const long SIZE = 7;
10     int *A = new int[SIZE];
11     int *a, *b, *B;
12
13     cout << "Input: ";
14     for (int i = 0; i < SIZE; i++)
15         cin >> A[i];
16
17     int n = *(A + 2);
18     if (n > 0)
19         B = fnc1(A, SIZE);
20     else
21         B = fnc2(A, SIZE);
22
23     displayArray(A, SIZE);
24     displayArray(B, SIZE);
25
26     a = B;
27     cout << *a << endl;
28     a += 4;
29     cout << *a << endl;
30     cout << ++(*a) << endl;
31     a++;
32
33     b = A + 1;
34     cout << *b << endl;
35     cout << *b++ << endl;
36     cout << *b << endl;
37
38     delete[] A;
39     return 0;
40 }
41
42 int* fnc1(const int *a, int size) {
43     int *newArray = new int[size];
44     for (int i = 0; i < size; i++)
45         newArray[i] = *(a + (i + 1) % size);
46
47     return newArray;
48 }
49
50 int* fnc2(int *a, int size) {
51     int *p = a;
52     while (p < &a[size]){
53         if (*p % 2 == 0)
54             *p = *p / 2;
55         else
56             *p = *p * 3 + 1;
57         p++;
58     }
59
60     return a;
61 }
62
63 void displayArray(const int arr[], int size) {
64     for (int index = 0; index < size; index++)
65         cout << arr[index] << " ";
66     cout << endl;
67 }

```

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

```

-2 10 -1 2 -8 16 5
-2 10 -1 2 -8 16 5
-2
-8
-7
10
10
-1

```

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

```

3 5 7 9 11 13 15
5 7 9 11 13 15 3
5
13
14
5
5
7

```

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

Solution: Yes, if $n > 0$ and fct1 is called a new array is created and pointed to by B in the main which is not deleted.

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:

Solution:

```
1 2
4 5 6
0 0
1 1
2 2
3 3
0 4
1 0
7 11
1
4
11
5
4
0
1
```

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

Solution: Yes, there is a dynamic array declared in Thing2 that is not deleted. Thing2 needs a destructor.

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
```

Solution:

```

#include <iostream>
using namespace std;

class Simulator {
private:
    int maxrandom;
    int simlength;
    int *simulation;
    int *counts;

    void RunSimulation();
    void Clear();

public:
    Simulator(int sl = 100, int mr = 6);
    ~Simulator();
    Simulator(const Simulator &obj);
    const Simulator operator=(const Simulator &right);
    friend ostream& operator <<(ostream&, const
        Simulator&);
    void reset(int sl = 100, int mr = 6);
    void Stats();
};

Simulator::Simulator(int sl, int mr) {
    srand(time(0));
    maxrandom = mr;
    simlength = sl;
    simulation = new int[simlength];
    counts = new int[mr];
    Clear();
    RunSimulation();
}

Simulator::~Simulator() {
    delete[] simulation;
    delete[] counts;
}

Simulator::Simulator(const Simulator &obj) {
    maxrandom = obj.maxrandom;
    simlength = obj.simlength;
    simulation = new int[simlength];
    counts = new int[maxrandom];

    for (int i = 0; i < simlength; i++)
        simulation[i] = obj.simulation[i];

    for (int i = 0; i < maxrandom; i++)
        counts[i] = obj.counts[i];
}

const Simulator Simulator::operator=(const Simulator &
    right) {
    delete[] simulation;
    delete[] counts;

    maxrandom = right.maxrandom;
    simlength = right.simlength;
    simulation = new int[simlength];
    counts = new int[maxrandom];

    for (int i = 0; i < simlength; i++)
        simulation[i] = right.simulation[i];

    for (int i = 0; i < maxrandom; i++)
        counts[i] = right.counts[i];

    return *this;
}

void Simulator::RunSimulation() {
    for (int i = 0; i < simlength; i++)

        simulation[i] = rand() % maxrandom + 1;

    for (int i = 0; i < simlength; i++)
        counts[simulation[i] - 1]++;
}

void Simulator::Clear() {
    for (int i = 0; i < simlength; i++)
        simulation[i] = 0;

    for (int i = 0; i < maxrandom; i++)
        counts[i] = 0;
}

ostream& operator<<(ostream &strm, const Simulator &
    obj) {
    strm << "Simulation: ";
    for (int i = 0; i < obj.simlength; i++)
        strm << obj.simulation[i] << " ";
    strm << endl;

    strm << "Counts: ";
    for (int i = 0; i < obj.maxrandom; i++)
        strm << obj.counts[i] << " ";
    strm << endl;

    return strm;
}

void Simulator::reset(int sl, int mr) {
    delete[] simulation;
    delete[] counts;

    maxrandom = mr;
    simlength = sl;
    simulation = new int[simlength];
    counts = new int[maxrandom];
    Clear();
    RunSimulation();
}

void Simulator::Stats() {
    cout << "Simulation Length: " << simlength << endl
        << endl;
    cout << "Simulation Range: 1 to " << maxrandom <<
        endl;
    int maxval = -1;
    for (int i = 0; i < maxrandom; i++)
        if (counts[i] > maxval)
            maxval = counts[i];

    int minval = maxrandom + 1;
    for (int i = 0; i < maxrandom; i++)
        if (counts[i] < minval)
            minval = counts[i];

    cout << "Maximum count is: " << maxval << " which
        occurs at value(s) ";
    for (int i = 0; i < maxrandom; i++)
        if (counts[i] == maxval)
            cout << (i + 1) << " ";

    cout << endl;

    cout << "Minimum count is: " << minval << " which
        occurs at value(s) ";
    for (int i = 0; i < maxrandom; i++)
        if (counts[i] == minval)
            cout << (i + 1) << " ";
    cout << endl;
}

```