1 Short Answer (5 Points Each)

1. Find and correct the errors in the following segment of code.

```
int x, *ptr = nullptr;
*ptr = &x;
```

Solution:

```
int x, *ptr = nullptr;
ptr = &x;
```

2. Find and correct the errors in the following segment of code.

```
int numbers[] = {10, 20, 30, 40, 50};
cout << "The third element in the array is ";
cout << *numbers + 3 << endl;
```

Solution:

```
int numbers[] = {10, 20, 30, 40, 50};
cout << "The third element in the array is ";
cout << *(numbers + 3) << endl;
```

3. Given the following code,

```
int *pint = nullptr;
pint = new int[10000];
```

What command will free the memory allocated by this segment?

Solution:

```
delete [] pint;
```

4. Write a function that will take as input a pointer to an integer array and the size of the array and return a pointer to a duplicate of the input array. The input array is not to be altered by the function. In this function, all array access is to be done with pointer references and dereferences, no bracket notation for any array access.

Solution:

```
int *duplicateArray(const int *arr, int size)
{
    int *newArray = nullptr;

    if (size <= 0)
        return nullptr;

    newArray = new int[size];

    for (int index = 0; index < size; index++)
        *(newArray + index) = *(arr + index);

    return newArray;
}
```

5. Find and correct the errors in the following segment of code.

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

Class Moon;
{
  Private;
    double earthWeight;
    double moonWeight;
  Public;
    moonWeight(double ew);
      { earthWeight = ew; moonweight = earthWeight / 6; }
    double getMoonWeight();
      { return moonWeight; }
}
```

Solution:

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

class Moon
{
  private:
    double earthWeight;
    double moonWeight;
  public:
    void setEarthWeight(double ew)
      { earthWeight = ew; moonWeight = earthWeight / 6; }
    double getMoonWeight()
      { return moonWeight; }
};
```

6. 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.

   (a) When an object of the class is returned by value: return obj;.

   (b) When an object of the class is passed to a function by value as an argument: fct(MyObj obj).

   (c) When an object is constructed based on another object of the same class: MyObj obj = obj2.

   (d) When compiler generates a temporary object.

7. When are copy constructors and overloaded assignment operators necessary?

   Solution: When there are data members of a class that are pointers.

8. Say we have a class named Thing, what is the specification for overloading its postfix ++ operator?

   Solution:

   ```
   Thing operator ++ (int);
   ```

9. Why would a programmer want to overload operators rather than use regular member functions to perform similar operations?

   Solution: Code reuseability. If operators for a class are overloaded then the same code that works on native data types will also work on the class structure without the need to rewrite the code to use member functions.

2 Program Trace (15 Points)

Write the output of the following program.

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

Output

Solution:

```
1 2 3 4 5 6 7
1 4 7 3 6 2 5
1
7
8
1 4 8 3 6 2 5
5
5
6
4
8    4
```

3 Coding (10 Points Each)

Given the following specification for the IntegerList class, write the following functions of the class as if these were contained in a separate IntegerList.cpp file.

```
1  class IntegerList
2  {
3    private:
4      int *list;
5      int numElements;
6      bool isValid(int) const;
7
8    public:
9      IntegerList(int sz = 1);
10     ~IntegerList();
11     void setElement(int, int);
12     int getElement(int) const;
13     void displayList() const;
14
15     IntegerList(const IntegerList &obj);
16     const IntegerList operator+(const IntegerList &right);
17     void resize(int);
18     const IntegerList sublist(int, int);
19 };
```

1. Write the constructor and the destructor.

   Solution:

```
IntegerList::IntegerList(int size)
{
    list = new int[size];
    numElements = size;
    for (int ndx = 0; ndx < size; ndx++)
        list[ndx] = 0;
}

IntegerList::~IntegerList()
{
    delete [] list;
}
```

2. Write the copy constructor.

   Solution:

```
IntegerList::IntegerList(const IntegerList &obj)
{
    numElements = obj.numElements;
    list = new int[numElements];
    for (int i = 0; i < numElements; i++)
        list[i] = obj.list[i];
}
```

3. Write the overloaded + operator. The overload of the + operator is to concatenate the two lists. So if l1, l2, and l3 are lists, the command `l3 = l1 + l2;` will assign l3 the combined list of l1 and l2.

   Solution:

```
const IntegerList IntegerList::operator+(const IntegerList &right)
{
    IntegerList newList(numElements + right.numElements);

    for (int i = 0; i < numElements; i++)
        newList[i] = list[i];

    for (int i = 0; i < right.numElements; i++)
        newList[i + numElements] = right.list[i];

    return newList;
}
```

4. Write the resize function. The resize function resizes the array keeping the contents of the array. If the new array size is smaller the data at the end of the old array will be lost. If the new array size is larger then the extra entries are to be set to 0.

   Solution:

```
1  void IntegerList::resize(int sz)
2  {
3      int *newArray = new int[sz];
4
5      for (int i = 0; i < sz; i++)
6          newArray[i] = 0;
7
8      int min = sz;
9      if (min > numElements)
10         min = numElements;
11
12     for (int i = 0; i < min; i++)
13         newArray[i] = list[i];
14
15     numElements = sz;
16     delete [] list;
17     list = newArray;
18 }
```

5. Write the sublist function. This function will return a new IntegerList of the segment between the first and second input positions, inclusively. So the call to `sublist(5, 17)` will return a new IntegerList object that has 13 elements in it, which are the values between the 5 and 17 positions of the original list. If the bounds are not in the correct order, reverse them. Also, make sure that beginning and ending indexes are handled, index less than 0 is set to 0 and an index larger then the maximum of the array is reset to the maximum of the array.

   Solution:

```
1  const IntegerList IntegerList::sublist(int b, int e)
2  {
3      if (b > e)
4      {
5          int temp = b;
6          b = e;
7          e = temp;
8      }
9
10     if (b < 0)
11         b = 0;
12
13     if (e < 0)
14         e = 0;
15
16     if (b >= numElements)
17         b = numElements - 1;
18
19     if (e >= numElements)
20         e = numElements - 1;
21
22     IntegerList newList(e - b + 1);
23     for (int i = 0; i < e - b + 1; i++)
24         newList[i] = list[b + i];
25
26     return newList;
27 }
```