Name _____

CISC 3142 – Programming Paradigms in C++
Spring 2019
Exam 2

1. a. Write a function that accepts an array and a size and prints out the elements of the array. You may only use subscript notation. **(5 points)**

```
void print(int arr[], int n) {
      for (int i = 0; i < n; i++)
            cout << att[i] << " ";
}
```

b. Same as *part a*, but now you may only use pointer notation (for any declarations as well as the code) and a while loop **(5 points)**

```
void print(int *arr, int n) {
        int *p = arr;
        while (p < arr + n)
                cout << *p << " ";
}
```

2. The notation `new int(17)` (note the parens and not []'s) means "dynamically allocate an integer and initialize it to `17`).

   Given the following declarations:

```
int *ip = new int(17);
int *jp = new int(23);
```

   a. Write a fragment of code that swaps the values pointed at by `ip` and `jp`, so that afterwards, the value pointed at by `ip` is `23` and that pointed at by jp is `17`. `ip` and `jp` should still be pointing at the same memory locations as before **(5 points)**

```
int t = *ip
*ip = *jp;
*jp = t;
```

   b. Now write a fragment in which the pointers should be swapped, i.e., afterwards, `ip` shold be pointing at the location `jp` was pointing at, and vice versa **(5 points)**

```
int *tp = ip;
ip = jp;
jp = tp;
```

3. Here are several implementations of `checkCapacity`; state what is wrong (or 'bad') with each of them (make sure you look at the parameter declarations as well; also, there is only basic thing wrong with each implementation). Do not simply point at the offending code—you MUST explain what the issue is. **(20 points)**

a.

```
void checkCapacity(int *&arr, int size, int &capacity) {
   if (size == capacity) {
           capacity *= 2;
           delete [] arr;
           int *tmp = new int[capacity];
           for (int i = 0; i < size; i++)
                   tmp[i] = arr[i];
           arr = tmp;
   }
}
```

The original array is deleted prior to copying its contents

b.

```
void checkCapacity(int *&arr, int size, int &capacity) {
   if (size == capacity) {
           capacity *= 2;
           int *tmp = new int[capacity];
           for (int i = 0; i < size; i++)
                   tmp[i] = arr[i];
           arr = tmp;
   }
}
```

The original array is not deleted

c.

```
void checkCapacity(int *&arr, int size, int &capacity) {
   if (size == capacity) {
           capacity *= 2;
           int *tmp = new int[capacity];
           for (int i = 0; i < capacity; i++)
                   tmp[i] = arr[i];
           delete [] arr;
           arr = tmp;
   }
}
```

The new capacity is used for the number of elements to copy – will go beyond the bounds of the original array

d.

```
void checkCapacity(int *&arr, int size, int &capacity) {
   if (size == capacity) {
           capacity++;
           int *tmp = new int[capacity];
           for (int i = 0; i < size; i++)
                   tmp[i] = arr[i];
           delete [] arr;
           arr = tmp;
   }
}
```

Simply increasing the capacity by one each time causes constant resizing (this is the 'bad' version.

e.

```
void checkCapacity(int *arr, int size, int capacity) {
   if (size == capacity) {
           capacity *= 2;
           int *tmp = new int[capacity];
           for (int i = 0; i < size; i++)
                   tmp[i] = arr[i];
           delete [] arr;
           arr = tmp;
   }
}
```

The array and capacity parameters are passed by value, rather than by reference, so the changes made to them do not persist upon returning from the function

4. Recall the state (data members) of the dynamically-sized array/vector class:

```
class Array {
     …
private:
     int size, capacity;
     int *arr;
}
```

a. Why is this class a candidate for the canonical form? What would happen if the canonical form was NOT used for this class? **(5 points)**

It has a pointer to data that will be dynamically allocated (and 'owned' by the object).

Not having the canonical form will cause the default behavior of a shallow rather than a deep copy, resulting in aliasing (reference rather than copy semantics)

b. What functions are <u>required</u> by the canonical form? **(5 points)**

Copy constructor
Destructor
Assignment operator

5. a. **(15 points)** Write a class template named `Statistician`, whose purpose is to allow one to keep track of various statistics regarding values submitted to an object of the class. In particular, the following behavior should be supported:

- A default constructor (which means initially there are no values passed to the class)
- A constructor that accepts a single value which becomes the first value passed to the class
- A constructor that accepts an array of values, and a size. These values become the first elements submitted to the class
  - (Feel free to combine any/all of the above constructors)
- A `+=` operator that accepts a value (if you have problems with the operator part, make it an `add` function).
- A `getMax` function that returns the largest value submitted so far
- A `getNumValues` function that returns the number of values submitted
- An insertion (`<<`) operator that prints out the max value and number of values submitted.

```
template <typename T>
class Statistician {
public:
        Statistician() : numValue(0), {}
          Statistician(const T &t) : numValues(0) {*this += t;}    // or add(t)
          Statistician(T *arr, int n) : numValues(0) {
              for (int i = 0; i < n; i++)
                      *this += arr[i];                          // or add(t)
          }
          Statistician &operator += (const T &t) {
              numValues++;
          if (t > maxValue) maxValue = t;
          }
          T &getMax() {
              If (numValues == 0) throw ….    // no max yet
          return maxValue;
          }
          int getNumValues() {return numValues;}
private:
        T maxValue;
        int numValues;
};
```

b. Could any type be used for instantiating a `Statistician`? If not, describe what would prevent instantiation of a type. **(5 points)**

Need whatever comparison operator/function used to calculate the max (I used >)

c. Provide some illustrative code using a type suitable for instantiation. (don't get too fancy; bare bones is fine—declare the object, add a couple of values, then call getMax and print the object) **(5 points)**

```
Statistician<int> si(10);
si += 12;
si += 3;
cout << si << " " << si.getMax() << " " << si.getValues() << endl/
```

(Any) one or more of the constructors is fine

d. What is the problem with using the default constructor and immediately calling `getMax`? How might you handle that situation? **(5 points)**

No max when 0 submissions… simplest is to do what I did, throw an exception.

e. How would adding a `getAverage` method (with the obvious semantics) change the set of legal instantiation types? Can you think of a type that would no longer be legal? **(5 points)**

Would also require an '/' operator. string would no longer be acceptable (unless is has a / operator, which in any event would probably not produce what we think of as an 'average'

6. Write a function template named `contains` that accepts an array, a size, and a value and returns whether the value appears in the array. **(5 points)**

```
template <typenameT>
bool contains(T *arr, int n, const T &val) {
        for (int I = 0; I < n; i++)
                if (arr[i] == val) return true;
        return false;
}
```