Lab 1: Data Sets (100 pts)

Introduction:

In math, a Set is a collection of elements that admits no duplicates. For example {1, 2, 3} is a Set while {1, 3, 2, 3} is not. Notice that the order of the elements does not matter {1, 2, 3} = {3, 2, 1}. Like numbers, Sets can have binary options … namely:

Union (U): union(A, B) – defined as all elements in A and B together, without duplicates.

{3, 4, 1} U {5, 4, 8} = {3, 4, 1, 5, 8}

Intersection (∩): intersection(A, B) – defined as all the elements common to both A and B.

{3, 4, 1} ∩ {5, 4, 8} = {4}

Difference (-): difference(A, B) – defined as the set of elements in A, which are not present in B; thus, what makes A different to B.

{3, 4, 1} - {5, 4, 8} = {3, 1}

What you have:

DArray.h: this file, shown in class, contains a library for working with dynamic arrays. The dynamic array will prove to be a great structure for building up our DataSet.h structure, which in a sense is just a dynamic array that allows no duplicates inside.

main.c: A test program that checks for all the functions that you are required to write. Below my output of main.c so that you know what your answers must be.

Errors that you might find:

“dereferencing pointer to incomplete type”: make sure you are including DArray.h inside of DataSet.h. This happens when the library is not there, therefore, the compiler does not know what someSet->array might be.

“conflicting types for … som function name”: remember what we said about functions and main. Before using a function, its caller (in our class example main) has to have function prototype defined. Either that or place the function definition before you use it in the code. When we divide our code into .h and .c we won’t have this problem.

Sample Output:

```
Warning: Size of dynamic array is increasing to 20
Warning: Size of dynamic array is increasing to 20
Warning: Size of dynamic array is increasing to 20
Warning: Size of dynamic array is increasing to 40
Warning: Size of dynamic array is increasing to 20
Warning: Size of dynamic array is increasing to 20
Warning: Size of dynamic array is increasing to 40
SET A IS:
Dynamic Array Info:
Length = 16
Size = 20
Data = {0, 19, 16, 5, 1, 17, 10, 14, 12, 6, 11, 7, 4, 8, 13, 9}
SET B IS:
Dynamic Array Info:
Length = 19
Size = 20
Data = {0, 16, 47, 10, 38, 43, 41, 12, 6, 11, 7, 48, 17, 33, 42, 36, 1, 4, 8}
A UNION B:
Dynamic Array Info:
Length = 24
Size = 40
Data = {0, 19, 16, 5, 1, 17, 10, 14, 12, 6, 11, 7, 4, 8, 13, 9, 47, 38, 43, 41, 48, 33, 42, 36}
A INTERSECTION B:
Dynamic Array Info:
Length = 11
Size = 20
Data = {0, 16, 1, 17, 10, 12, 6, 11, 7, 4, 8}
A DIFF B:
Dynamic Array Info:
Length = 5
Size = 10
Data = {19, 5, 14, 13, 9}
ORIGINAL SET:
Dynamic Array Info:
Length = 24
Size = 40
Data = {0, 19, 16, 5, 1, 17, 10, 14, 12, 6, 11, 7, 4, 8, 13, 9, 47, 38, 43, 41, 48, 33, 42, 36}
SUBSET:
Dynamic Array Info:
Length = 5
Size = 10
Data = {5, 1, 17, 10, 14}
subsetTest IS a subset of unionSet
unionSeet IS NOT a subset of subsetSet
THE EMPTY SET LOOKS LIKE:
Dynamic Array Info:
```

```
Length = 0
Size = 10
Data = {}
emptySet IS equal to the NULL set
reverseSet IS equal unionSet
reverseSet CONTAINSS 0
Dynamic Array Info:
Length = 24
Size = 40
Data = {36, 42, 33, 48, 41, 43, 38, 47, 9, 13, 8, 4, 7, 11, 6, 12, 14, 10, 17, 1, 5, 16, 19, 0}
ORGINAL SET FOR STATISTICS:
Dynamic Array Info:
Length = 24
Size = 40
Data = {0, 19, 16, 5, 1, 17, 10, 14, 12, 6, 11, 7, 4, 8, 13, 9, 47, 38, 43, 41, 48, 33, 42, 36}
Min value 0
Max value 48
Avg value 20.000000
Range value 48
```

Submission:

Submit the DataSet.h file that you will be developing. You do not need to submit anything else, as we have main.c and DArray.h. Below some requirements for DataSet.h.

Requirements:

The purpose of the first lab will be to get started with your Eclipse development environment, as well understanding/using someone else code, to write your own. You are provided/allowed to use the file named DArray.h, which holds the struct DArray. Your tasks are as follow (5 points each):

1. File header (Names of pair/team, course name and term, assignment name, purpose) and comments describing the code.
   This step simply asks for commenting the code that you write in DataSet.h. The file header is a block of comments that identifies you and your partner as the authors who will receive credit for this code.

   Example for the DataSet.h file, the only file you have to work on, and that is created on step 3.

   //File: DataSet.h (step 3)

   /*Step 1

   Author: Hector M Lugo-Cordero

   Class: COP3502C, SummerC 2014

   DataSet.h: This file implements a math set and its operations. A math set is defined as linear collection elements with no duplicates on it. This library depends on the DArray.h as the internal array of the DataSet is really a DArray with additional functionality to ensure that no duplicates are found and the operations are possible.

   */

2. Sample output in debug mode
   In this step you need to take a screenshot (of the command prompt preferred) of your final work, once main.c runs. If everything is correct, the content of the screenshot will be equivalent to the sample output provided earlier.

3. Create a library file named DataSet.h, you can use DArray.h as a template of what the DataSet.h should contain.
   Here is where you create the DataSet.h file that you will submit. Steps 1 and 2 will be common for all lab assignments, that's why they were placed at the beginning of the steps. Your Eclipse

IDE will most likely create a header file named DataSet.h with the following content at the beginning.

```
#ifndef DATASET_H_

#define DATASET_H_

//Here is where you will work your lab (THIS IS NOT ADDED BY ECLIPSE, I typed this; on Eclipse it will be empty)

#endif /* DATASET_H_ */
```

4. The DataSet should hold a Dynamic Array at its core; in addition the set should have information about its minimum value and the maximum value.

   I am helping out so that everyone has a good start on their first lab … this step is asking to create the struct that will hold our DataSet, just like the one I had for DArray:

   ```
   struct DataSet{

      struct DArray* array;

      Byte min;

      Byte max;

   };
   ```

   Just make sure that the main.c compiles, so the names in your DataSet.h match those required by main.c

5. Alloc – struct DataSet* allocDataSet()
   This function allocates the data, for the first lab I am also providing it:
   //step 5: you do not need a size as the only thing you need to allocate is the DArray. I know, DArray needs a size, but just did what I did on the previous main discussed in class, pass 10 or any other number. Its Dynamic after all, so it will increase its size once it needs to.

   ```
   struct DataSet* allocDataSet(){
      struct DataSet* dataSet = (struct DataSet*)malloc(sizeof(struct DataSet);
      dataSet->setData = allocDArray(10);    //code reuse
      //oposite extremes are assigned to one another such that they may be updated when some
   entered value is smaller than minValue or larger than maxValue
      dataSet->minValue = 255;
      dataSet->maxValue = 0;
     return dataSet;
   ```

}

6. release –void releaseDataSet(struct DataSet* set)
   This function frees the memory space allocated earlier by allocDataSet. Remember to free any pointer that DataSet holds before freeing the DataSet itself. Hint: use the releaseDArray function.

7. Append function: add information at the end, keeping always the set property valid (no duplicates).

   //step 7 (Hint: use the contains function to know if you need to add or not an element to the DataSet. I believe DArray already has one, if not you can always write the one being asked in step 19 and call it from this one)

   void appendDataSet(struct DataSet* dataSet, Byte newValue){ }

8. Set union: struct DataSet* unionDataSet(struct DataSet* A, struct DataSet* B) – returns all elements in A along with those in B, with no duplicates.
   See example at the start of the document

9. Set intersection: struct DataSet* intersectionDataSet(struct DataSet* A, struct DataSet* B) – returns all elements in both A and B.
   See example at the start of the document

10. Set difference: struct DataSet* diffDataSet(struct DataSet* A, struct DataSet* B) – returns all elements in A only (not in B)
    See example at the start of the document

11. struct DataSet* subset(struct DataSet* set, uint start, uint end) – creates a subset of the set from position start to end

    //step 11: return a DataSet that contains all elements on mainSet[start:end]. That is the elements on the mainSet, starting at start position until end position. Thus, the result is a subset of mainSet

    struct DataSet* subset(struct DataSet* mainSet, int start, int end){ }

12. bool isSubset(struct DataSet* A, struct DataSet* B) – returns true if all elements in A are found in B, meaning that A is a subset of B.

    //step 12: must include stdbool.h ... returns true if all elements in A are found in B and A is smaller than B, meaning that A is a subset of B

    bool isSubset(struct DataSet* A, struct DataSet* B){   }

13. float Average(struct DataSet* set) – returns the average value inside the set of values hold by the DataSet

14. Word Sum(struct DataSet* set) – adds all the elements located inside the DataSet. Could help for step 13.

15. struct DataSet* Square(struct DataSet* set) – returns a new set with all elements squared
    //step 15: Notice that you could use the applyDArray function to just apply a pow function or a square function defined by you (there is one located at main). The result is returned as a DataSet and not as a Word*, meaning 2 things: 1) the squared values will be the elements of a dataset and 2) some or maybe all values will overflow the maximum value of a Byte. This is for you to see what happens that occurs. I will provide with what the answer to the whole lab is so that you can compare with yours.

16. Byte range(struct DataSet* set) – returns the range of the set, i.e., |max – min|
17. void printDataSet(struct DataSet* dataSet) – Print the set
18. bool isNull(struct DataSet* set) – returns true if the set is a null set
    //step 18: returns true if testSet is the NULL set, meaning it empty with no elements inside of it, useful specially when you want to check if the intersection of two sets is empty

19. bool contains(struct DataSet* set, Byte val) – returns true if an element val is contained in the set
    DArray already has this function, if you are using DArray as your container for the set, then you can call its contains from this function too.

20. bool equals(struct DataSet* A, struct DataSet* B) – returns true if both sets have the same values

    //step 20: returns true if all elements in A are found in B and they both have the same length. The order of the elements is not important for testing equality of two sets.