Project: Dynamic Array Sets

In this project, you will create functions that work with mathematical sets stored in dynamic arrays.

# About Sets

A mathematical set is an unordered collection of items without duplicates. The items in the set are referred to as elements. The elements are typically written in curly braces separated by commas. For instance, a set containing the elements a, b, and c would typically be written:

{a, b, c}

Since the order of the set doesn’t matter, this set is equivalent to

{b, a, c}

Sets can be empty (ie contain no elements). The empty set is typically written as a set of curly braces with nothing inside:

{}

For this project, set elements are stored in dynamic arrays of characters. For instance, the set {a, b, c} could be created with the code:

int setSize = 3;

char\* aSet = new char[setSize];

aSet[0] = ‘a’;

aSet[2] = ‘b’;

aSet[3] = ‘c’;

An empty set (ie set with no elements) would be:

int setSize = 0;

char\* aSet = nullptr; // no elements, so no array

As with all arrays in C++, since there is no convenient way to determine the number of elements in a dynamic array, **the current size of all dynamic arrays must be tracked with a variable**.

# General Tips

Don’t try to write all of this at once! Start with a small set of core functions, test and get those working, then progressively add other pieces. Though it won’t be the case for all projects, I recommend working through the required list of functions from top to bottom below.

Don’t duplicate code! If you have a function that does what you need to do (such as addElement()) then call that function!

Watch out for code that tries to access invalid array indices (i.e. out of array bounds)! It’s very easy to do.

# General Requirements

You will work with 3 files for this project:

* setFunctions.h – This file contains the function declarations for the required functions listed below. The only thing you need to do in this file is add Javadoc style function comments. You may want to do this last after you have a thorough understanding of the functions. See the course style guide for details on Javadoc style function comments.
* setFunctions.cpp – This is the file where you will need to provide the function definitions (see details below).
* driver.cpp – This file includes main() which runs a series of tests of the required functions. You are encouraged to edit this file and you do not need to turn it in. You most likely will *not* want all the tests running when you begin working on the project. Comment out the tests you don’t want running, then restore them later. You may also want to change the existing tests or add new ones.

# Required Functions

| bool containsElement(char\* set, int size,  char itemToCheck);  Given a pointer to a dynamic array containing a set (set), the number of elements in the set (size), and an element (itemToCheck), the function returns true if the set contains the given element. |
| --- |
| void display(char\* set, int size);  Given a pointer to a dynamic array containing a set (set), the number of elements in the set (size), displays the elements of the array in set format.  For instance, if the dynamic array contains   | a | b | c | | --- | --- | --- |   The set should be displayed as:  {a, b, c}  This should work correctly with an empty set (ie set is nullptr) and display:  {}  This function should end its output with an endl. |
| char\* addElement(char\* set, int& size,  char itemToAdd , int& newSize);  Given a pointer to a dynamic array containing a set (set), the number of elements in the set (size), and an element (itemToAdd), the function forms a new set containing the elements of the original set plus itemToAdd if that element doesn’t already exist in the original set. The function returns a pointer to the new resulting set and updates the pass-by-reference variable newSize to match the size of the resulting set.  Since duplicate elements are never added, the resulting set may be identical to the original set passed in if itemToAdd already exists in the set. If itemToAdd is added to the set it should always be added as the last element (ie the highest index) in the newly formed set.  Note that there is no scenario where the dynamic array pointer passed in (set) is the same pointer that is returned, nor would the dynamic array pointed to by set ever be altered. In other words, this function always creates a new dynamic array and returns a pointer to it even if it is just a copy of the original array passed in.  For instance, if set points to a dynamic array containing the set  {a, b, c}  and itemToAdd is x, then newSize should be set to 4 and the function should return a pointer to a dynamic array containing the set  {a, b, c, x}  On the other hand, if set points to a dynamic array containing the set:  {n, e, t}  and itemToAdd is e, then newSize should be set to 3 and the function should return a pointer to a new dynamic array containing a copy of the original set:  {n, e, t} |
| char\* setUnion(char\* set1, int set1Size, char\* set2,  int set2Size, int& unionSize);  The union of 2 sets is defined as a new set containing all the elements of the original 2 sets. For instance, the union of sets {a, b, c} and {c, d, e} is a set of 5 elements: {a, b, c, d, e} (note there are no duplicates).  This function should return the pointer to a dynamic array containing the union of the set contained in the dynamic array pointed to by set1 with the set contained in the dynamic array pointed to by set2 and update unionSize to contain the number of elements in the union.  It is possible for the intersection to result in an empty set if the 2 original sets contain no elements (ie set1 and set2 are both nullptr). In this case, the pointer returned should be nullptr and unionSize should be zero.  Neither of the original 2 sets passed in should be altered by this process.  The order of the items in the resulting set **does not matter**. |
| char\* setIntersect(char\* set1, int set1Size,  char\* set2, int set2Size, int& intersectSize);  The intersection of 2 sets is defined as a new set containing all elements that appear in both of the original 2 sets. For instance, the union of sets {a, b, c, d} and {c, d, e, f} is {c, d} (note there are no duplicates).  This function should return the pointer to a dynamic array containing the intersection of the set contained in the dynamic array pointed to by set1 with the set contained in the dynamic array pointed to by set2 and update intersectSize to contain the number of elements in the intersection.  It is possible for the intersection to result in an empty set if the 2 original sets contain none of the same elements. In this case, the pointer returned should be nullptr and intersectSize should be zero.  Neither of the original 2 sets passed in should be altered by this process.  The order of the items in the resulting set **does not matter**. |

# Turn In

Upload setFunctions.h and setFunctions.cpp prior to the due date.

Jacob Kier

Johnson County Community College