Sets
Recall that a set is just a collection of objects. The order in which the objects are listed doesn’t matter. So the set \( A = \{3, 1, 2\} \) is the same as the set \( B = \{2, 3, 1\} \). Repetitions also don’t matter. So the sets \( A \) and \( B \) are the same as the set \( C = \{3, 1, 2\} \). The union of two sets \( D \) and \( E \) is the set whose elements are the elements that belong to \( D \), to \( E \), or to both \( D \) and \( E \). For example, if \( D = \{1, 3, 5, 7\} \) and \( E = \{1, 2, 3, 4\} \), then their union is

\[
D \cup E = \{1, 3, 5, 7\} \cup \{1, 2, 3, 4\} = \{1, 2, 3, 4, 5, 7\}.
\]

The intersection of two sets \( D \) and \( E \) is the set of elements that they have in common. So the intersection of the sets \( D \) and \( E \) is

\[
D \cap E = \{1, 3, 5, 7\} \cap \{1, 2, 3, 4\} = \{1, 3\}.
\]

The difference of two sets \( D \) and \( E \) is the set of elements belonging to the first, but not the second. So the difference of the sets \( D \) and \( E \) is

\[
D - E = \{1, 3, 5, 7\} - \{1, 2, 3, 4\} = \{5, 7\}.
\]

Program 2
For programming assignment 2, you should write a C program that implements a set abstract data type with operations union, intersection and set difference. Input to the program will be a sequence of single characters separated by white space. These indicate which operations should be carried out:

- ‘u’ or ‘U’: Find the union of two sets,
- ‘i’ or ‘I’: Find the intersection of two sets, and
- ‘d’ or ‘D’: Find the difference of two sets.
- ‘q’ or ‘Q’: Quit (freeing any allocated memory),

For the union, intersection, or difference, the user will enter two sets \( A \) and \( B \), and your program should output the result of the operation on the sets. The elements of the sets will be nonnegative ints, \textit{listed in increasing order, without repetitions}. The last element in an input set will be followed by a negative value. For example, the set \( D = \{1, 3, 5, 7\} \) would be entered as
You can assume that input sets will be sorted lists of nonnegative ints in increasing order with no duplicates, and each input set will be terminated by a negative value. So you don’t need to check the input sets for correctness.

**Implementation Details**

Each set should be implemented as a sorted, singly linked list of ints, and there should be no repeated values in the stored linked list. So \( D \) should be stored as

\[
\text{head}_p \rightarrow 1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \ \backslash
\]

(The backslash (\) denotes the NULL pointer.) The set \( D \) should *not* be stored as

\[
\text{head}_p \rightarrow 3 \rightarrow 1 \rightarrow 7 \rightarrow 5 \ \backslash
\]
or

\[
\text{head}_p \rightarrow 1 \rightarrow 3 \rightarrow 3 \rightarrow 5 \rightarrow 7 \ \backslash
\]

Note that the restriction that the lists are sorted and that they don’t store duplicates doesn’t mean that we can’t accurately represent sets: any set of nonnegative ints can have its elements listed in increasing order with no repeated values. The purpose of the restrictions is to reduce the complexity of the program’s internal storage, *and* to make it possible to efficiently implement the operations.

Since the sets are stored in sorted order, all three operations can be implemented as variations on the merge operation. Recollect that we merge two sorted lists by comparing the elements of the list pairwise and appending the smaller element to the new list. If the input lists are \( A \) and \( B \), and the merged list is \( C \), we can define pointers \( a_p \), \( b_p \) and \( c_p \) that reference the current element in each list, respectively. So the merge operation proceeds as follows:

```
while (a_p != NULL && b_p != NULL)
    if (a_p->data <= b_p->data) {
        c_p = Append(&C, c_p, a_p->data);
        a_p = Advance(a_p);
    } else { // b_p->data < a_p->data
        c_p = Append(&C, c_p, b_p->data);
        b_p = Advance(b_p);
    }

while (a_p != NULL) {
    c_p = Append(&C, c_p, a_p->data);
    a_p = Advance(a_p);
```
while (b_p != NULL) {
    c_p = Append(&C, c_p, b_p->data);
    b_p = Advance(b_p);
}

You should use this algorithm as the basis of your implementations. Note that at no point do you sort any of the lists!

Each of the set operations, \( A \cup B \), etc., should be implemented by a function. These functions should not change the input arguments \( A \) and \( B \). If the result of the operation (e.g., \( C \)) is passed in to the function (instead of being created and returned by the function), then, of course, the function can change it. Note that none of the functions implementing the operations should do any I/O (except for debug I/O). The results should be printed by the calling function (e.g., \texttt{main}).

Since each new operation in the program reads in new sets, your program won’t be reusing the elements of the sets involved in the previous operation. So if you don’t free the nodes in each of the lists after printing the results of an operation, your program will have a memory leak: memory allocated from the heap will be lost.

**Development**

You should compile and test your program using Linux, since the parallel computers we have access to all use Linux (rather than Windows or MacOS X).

Working with linked lists and pointers can be confusing. You should be sure to develop your algorithms on paper before coding: check for different input and output conditions (e.g., one or more of the sets sets is empty).

Get your input and output functions fully debugged before starting implementation of the set operations. Clearly, you won’t be able to tell if your operations are correct if you don’t know what their input or their output is.

If your program is crashing or producing incorrect results, you can use the techniques we discussed in class for trying to determine the problem: careful use of \texttt{printf} and \texttt{fflush} and/or use of the debugger \texttt{gdb}.

**Extra Help**

I have office hours Mondays and Fridays from 4:45 to 5:45 and Wednesdays from 10:30 to 11:30. Erika has office hours on Thursdays from 3 to 4, and Mark has office hours on Thursdays from 4:30 to 5:30.

**Submission**

You should develop your program in the \texttt{p2} subdirectory of your local SVN tree. After you first create your source file, type
$ svn add sets.c
$ svn commit sets.c -m "create program 2 source file"

It's a good idea to frequently update this by typing something like

$ svn commit sets.c -m "modified program to do XXX"

This frequent updating is very important: if you accidentally corrupt or destroy your copy of the program, you can always retrieve the most recent version from the SVN server.

Be sure to commit your final version of the source code by 6 pm on Monday, October 5.

$ svn commit sets.c -m "final testing completed"

Grading

1. Correctness will be 60% of your grade. Does your program find the correct result given correct input? Does it correctly implement each set as a sorted, singly linked list? Does it have any memory leaks? Does it use the merge operation as the basis of each of the set operations? Does it sort the lists (which it should not do)?

2. Documentation will be 20% of your grade. Does your header documentation include the author’s name, the purpose of the program, and a description of how to use the program? Are the identifiers meaningful? Are any obscure constructs clearly explained? Does the function header documentation explain the purpose of the function, its arguments and its return value?

3. Source format will be 10% of your grade. Is the indentation consistent? Have blank lines been used so that the program is easy to read? Is the use of capitalization consistent? Are there any lines of source that are longer than 80 characters (i.e., wrap around the screen)?

4. Quality of solution will be 10% of your grade. Are any of your functions more than 40 lines (not including blank lines, curly braces, or comments)? Are there multipurpose functions? Is your solution too clever – i.e., has the solution been condensed to the point where it’s incomprehensible? Do any of your functions traverse a list unnecessarily? For example, do you need to use an extra traversal to find the last node in a list?

Academic Honesty

Remember that you can discuss the program with your classmates, but you cannot look at anyone else’s pseudo-code or source code. (Of course, this includes code that you might find on the internet.)