Name **:** Mu In Nasif
Roll **:** 001910501036
Class **:** BCSE II
Sem **:** 3
Session : 2020-2021

Data Structures and Algorithms Assignment Set 2

Question - 3: Define an ADT for List. Write C data structure representation and functions for the operations on the List in a Header file with array as the base data structure. Write a menu-driven main program in a separate file for testing the different operations and include the above header file. Two data structures with and without using sentinels in arrays are to be implemented

Solution Approach :

A list is a sequence of (possibly duplicated) elements of same type (sequence implies there is a start element, end element and other elements will have a predecessor and successor). The very basic operations as can be defined on lists are insertion of elements, deletion of elements and access of elements (traversal). Since we are using arrays as a backing structure for lists (called array lists or vectors) the operations that are defined and implemented are as follows:
1. Obtain the length of the list (number of elements)
2. Read the list from either direction (start to end or in reverse)
3. Retrieve the ith element
4. Store a new value (overwrite) at i'th position
5. Insert a new element at ith position (we state/define that the old elements starting from position i to the end are shifted one position towards the end)
6. Delete an element at ith position (we state/define that the elements starting from position i+1 to the end are shifted one position towards the start)
7. Search the list for a specified given value (called the key)
8. Sort the list in some order on the value of the elements. For the given implementation of the sorting algorithm in this paper, we use selection sort which proceeds as follows: for ascending order, search the minimum element, put it into the 1st position, and then repeat the process considering the sublist formed from the 2nd position to the end, and so continue to put all elements at their correct positions. We chose selection sort as it is conceptually quite easy to understand (any other sorting algorithm can be used as desired).

For the given operations below, we will consider NIL to be a sentinel value indicating the end of the array. Such given operations can easily be converted to non-sentinel versions by holding the length of the list as a property of the list.

Pseudocode:
We call our such list objects backed by an array as vectors, which are nothing but arrays. Positions start from

1. Vector read (iteration/traversal) in either direction, store and retrieval operations are trivial.

```
vector_length(V) {//get length of V.
 i = 1
        while V[i] is not equal to NIL: i = i + 1
                return i – 1 //Length of vector is number of elements excluding NIL.
}

vector_traverse(V, direction)
{//traverse V in given direction.
        if direction is first to last {
                 i = 1
                while V[i] is not equal to NIL {
                        visit(V[i]) //Visit the element (traversal)
                                i = i + 1
                }
        } else {
                len = vector_length(V)
                 for i = len to 1 step -1 inclusive: visit(V)
         }
}

vector_insert(V, i, element) {
/*Insert element at position i. Assume 1 <= i <= length of V + 1 and V has space to hold the
extra element.*/
         j = vector_length(V)
         V[j + 2] = NIL //Assign proper terminator for vector
        while j >= i {
                 V[j + 1] = V[j] //Shift each towards the end
                j = j – 1
        }
        V[i] = element
}

vector_delete(V, i) {
//Delete element at position i. Assume 1 <= i <= length of V.
        j = i
        while V[j+1] is not NIL {
                V[j] = V[j + 1] //Shift each towards the start
                j = j + 1
```

```
    }
    V[j] = NIL //Terminate the vector
}

vector_search(V, element, direction) {
    /*Search V for element in the direction specified. Return suitable index.*/
    i = 1 idx = NIL //Store final index
    while V[i] is not NIL {
        if V[i] equals element {
            idx = i
            if direction is first to last: return idx //Exit on first occurrence in case of
            first-to-last search
        }
        i = i + 1
    }
    return idx //Return last occurrence in case of last-to-first search
}

vector_sort_ascending(V) {
    /*Sort the vector in ascending order, here we use selection sort; this assumes elements
    that can be compared*/
    len = vector_length(V)
    for i = 1 to len inclusive {
        Search for the minimum element between i to len inclusive, and store the index of
        such element in k
        Swap V[i] with V[k]
    }
}
```

Code: a2q3_vectorSentinel.h is the implementation which uses sentinels and a2q3_vectorNonSentinel.h is the implementation which does not use sentinels. a2q3_driver.c is the testing program for the non-sentinel version, which can be converted to test the sentinel version instead by defining the macro USE_SENTINELS at the top of the file. This can also be done during compilation by using the following:

1. On GCC:
   ```
   gcc -DUSE_SENTINELS <...rest options...> p3_test.c
   ```
2. On Clang:
   ```
   clang -DUSE_SENTINELS <...rest options...> p3_test.c
   ```
3. On MSVC:
   ```
   CL /DUSE_SENTINELS <...rest options...> p3_test.c
   ```
   All source code conforms to ANSI C89