Project 3 – Pseudocode

**DoublyLinkedList**

**isEmpty()**

return whether *first* is pointing to null

// end isEmpty()

**insertUpdate(Presidents object)**

make Link *current* point to *first*

initialize preceding node counter to 0

case based on *object*’s transaction code

case “A”

loop while true

if list is empty

call insertFirst() and insert object at front of list

break

// end if

else if *object*'s last name is identical to *current*'s last name

if *object*’s name is identical to *current*’s name

print formatted string and “Dupe Add Attempted”

break

// end if

else if *object*'s first name alphabetically earlier than *current*'s first name

make new link

if last link

call insertLast() and insert *object*

// end if

else // not last link

make *newLink* point to old *previous*

make old *previous* point to *newLink*

// end else

make *newLink* point to old *current*

make old *current* point to *newLink*

print formatted string with *object*’s data

break

// end else if

// end else if

else if *object*'s last name alphabetically earlier than *current*'s last name

make new link

if last link

call insertLast() and insert *object*

// end if

else // not last link

make *newLink* point to old *previous*

make old *previous* point to *newLink*

// end else

make *newLink* point to old *current*

make old *current* point to *newLink*

print formatted string with *object*’s data

// end else if

increment preceding node counter

move to next link

// end while loop

break

// end case “A”

case “D”

loop while true

if list was traversed and item wasn’t found

print formatted string and “Transaction Not Found”

break

// end if

else if item was found in list

if first item

make *first* point to old *next*

// end if

else // not first item

make old *previous* point to old *next*

// end else

if last item

make *last* point to old *previous*

// end if

else // not last item

make old *next* point to old *previous*

// end else

print formatted string with object’s data

break

// end else if

increment preceding node counter

move to next link

// end while loop

break

// end case “D”

default case

print formatted string and “Invalid Transaction Type”

break

// end default case

// end case

// end insertUpdate()

**insertFirst(Presidents object)**

make new link

if list is empty

make *last* point to *newLink*

// end if

else // list is not empty

make old *first* point to *newLink*

// end else

make *newLink* point to old *first*

make *first* point to *newLink*

// end insertFirst()

**insertLast(Presidents object)**

make new link

if list is empty

make *first* point to *newLink*

// end if

else // list is not empty

make old *last* point to *newLink*

make *newLink* point to old *last*

// end else

make *last* point to *newLink*

// end insertLast()

**deleteFirst()**

make Link *temp* point to *first*

if only one item in list

make *last* point to null

// end if

else // more than one item in list

make old *next* point to null

// end else

make *first* point to old *next*

return *temp*

// end deleteFirst()

**deleteLast()**

make Link *temp* point to *last*

if only one item in list

make *first* point to null

// end if

else // more than one item in list

make old *previous* point to null

// end else

make *last* point to old *previous*

return *temp*

// end deleteLast()

**displayList()**

call displayListForwards()

call displayListBackwards()

// end displayList()

**displayListForwards()**

print header

print formatted column headers “LAST NAME”, “FIRST NAME”, “STATE”

make Link *current* point to *first*

while *current* isn’t null

display Link at *current*

make *current* point to *next* link

// end while

// end displayListForwards()

**displayListBackwards()**

print header

print formatted column headers “LAST NAME”, “FIRST NAME”, “STATE”

make Link *current* point to *last*

while *current* isn’t null

display Link at *current*

make *current* point to *previous* link

// end while

// end displayListBackwards()

**PresidentsMDrvr**

**insert(Presidents object)**

assign *object* to index *nElems* of myPresidents array

increment number of array elements

// end insert()

**bubbleSortOnLastName()**

declare integer *out*

declare integer *in*

loop (set *out* to number of array elements - 1; while *out* greater than 1; decrement *out*)

loop (set *in* to 0; while *in* less than *out*; increment *in*)

if (*lastName* at index *in* of myPresidents array is greater than *lastName* at index *in*+1 of myPresidents array)

call swap(*in*, *in*+1)

// end if

else if (*lastName* at index *in* of myPresidents array is equal to *lastName* at index *in*+1 of myPresidents array AND *firstName* at *in* is greater than *firstName* at *in*+1)

call swap(*in*, *in*+1)

// end else if

// end inner loop

//end outer loop

// end bubbleSortOnLastName()

**swap(int one, int two)**

declare Presidents object *temp* and set to myPresidents[one]

set myPresidents[one] to myPresidents[two]

set myPresidents[two] to *temp*

// end swap()

**displayArray()**

print header

print formatted column headers “LAST NAME”, “FIRST NAME”, “STATE”

loop through myPresidents array

print each element

// end loop

// end displayArray()

**movePresidents(DoublyLinkedList list)**

loop through myPresidents array

if *party* of Presidents object at index *i* of myPresidents is “Republican”

call insertLast() and put object at end of list

// end if

// end loop

// end movePresidents()