Project 2 – Pseudocode

**Queue**

**insert(Presidents object)**

case based on party attribute of *object*

case “Democrat”

pre-increment *rearDem* pointer and add *object* to its index in *queueDem*

increment *nItemsDem*

break

// end case “Democrat”

case “Republican”

pre-increment *rearRepub* pointer and add *object* to its index in *queueRepub*

increment *nItemsRepub*

break

// end case “Republican”

case “Whig”

pre-increment *rearWhig* pointer and add *object* to its index in *queueWhig*

increment *nItemsWhig*

break

// end case “Whig”

// end case

// end insert()

**priorityInsert(Presidents object)**

declare integer *j*

case based on party attribute of *object*

case “Democrat”

if no items in Democrat array list

increment *nItemsDem* and add *object* to its index in *queueDem*

// end if

else // items in Democrat array list

loop (set *j* to *nItemsDem* - 1; while *j* is greater than or equal to 0; decrement *j*)

if number of *object* is less than or equal to number of element at position *j*

get element at position *j* and add it to position (*j*+1) in *queueDem*

remove element at position *j* from *queueDem*

// end if

else // greater than number of object at current index

break

// end else

// end loop

add *object* to position (*j*+1) in *queueDem*

increment *nItemsDem*

// end else

break

// end case “Democrat”

case “Republican”

if no items in Republican array list

increment *nItemsRepub* and add *object* to its index in *queueRepub*

// end if

else // items in Republican array list

loop (set *j* to *nItemsRepub* - 1; while *j* is greater than or equal to 0; decrement *j*)

if number of *object* is less than or equal to number of element at position *j*

get element at position *j* and add it to position (*j*+1) in *queueRepub*

remove element at position *j* from *queueRepub*

// end if

else // greater than number of object at current index

break

// end else

// end loop

add *object* to position (*j*+1) in *queueRepub*

increment *nItemsRepub*

// end else

break

// end case “Republican”

case “Whig”

if no items in Whig array list

increment *nItemsWhig* and add *object* to its index in *queueWhig*

// end if

else // items in Whig array list

loop (set *j* to *nItemsWhig* - 1; while *j* is greater than or equal to 0; decrement *j*)

if number of *object* is less than or equal to number of element at position *j*

get element at position *j* and add it to position (*j*+1) in *queueWhig*

remove element at position *j* from *queueWhig*

// end if

else // greater than number of object at current index

break

// end else

// end loop

add *object* to position (*j*+1) in *queueWhig*

increment *nItemsWhig*

// end else

break

// end case “Whig”

// end case

// end priorityInsert()

**selectionSortDem()**

declare integer *out*

declare integer *in*

declare integer *min*

outer loop (set *out* to 0; while *out* is less than *nItemsDem* - 1; increment *out*)

set *min* to *out*

inner loop (set *in* to *out* + 1; while *in* is less than *nItemsDem* - 1; increment *out*)

if number of element at index *min* of *queueDem* is greater than number of element at index *in* of *queueDem*

set *min* to *in*

// end if

call swapDem(*out*, *min*)

// end inner loop

// end outer loop

// end selectionSortDem()

**selectionSortRepub()**

declare integer *out*

declare integer *in*

declare integer *min*

outer loop (set *out* to 0; while *out* is less than *nItemsRepub* - 1; increment *out*)

set *min* to *out*

inner loop (set *in* to *out* + 1; while *in* is less than *nItemsRepub* - 1; increment *out*)

if number of element at index *min* of *queueRepub* is greater than number of element at index *in* of *queueRepub*

set *min* to *in*

// end if

call swapRepub(*out*, *min*)

// end inner loop

// end outer loop

// end selectionSortRepub()

**selectionSortWhig()**

declare integer *out*

declare integer *in*

declare integer *min*

outer loop (set *out* to 0; while *out* is less than *nItemsWhig* - 1; increment *out*)

set *min* to *out*

inner loop (set *in* to *out* + 1; while *in* is less than *nItemsWhig* - 1; increment *out*)

if number of element at index *min* of *queueWhig* is greater than number of element at index *in* of *queueWhig*

set *min* to *in*

// end if

call swapWhig(*out*, *min*)

// end inner loop

// end outer loop

// end selectionSortWhig()

**selectionSortAll()**

call selectionSortDem()

call selectionSortRepub()

call selectionSortWhig()

// end selectionSortAll()

**swapDem(int one, int two)**

declare Presidents object *temp* and set to element at index *one* of *queueDem*

get element at index *two* of *queueDem* and set it to element at index *one* of *queueDem*

set *temp* to element at index *two* of *queueDem*

// end swapDem()

**swapRepub()**

declare Presidents object *temp* and set to element at index *one* of *queueRepub*

get element at index *two* of *queueRepub* and set it to element at index *one* of *queueRepub*

set *temp* to element at index *two* of *queueRepub*

// end swapRepub()

**swapWhig()**

declare Presidents object *temp* and set to element at index *one* of *queueWhig*

get element at index *two* of *queueWhig* and set it to element at index *one* of *queueWhig*

set *temp* to element at index *two* of *queueWhig*

// end swapWhig()

**displayQueueDem()**

print “Queue 1: Democrats”

print formatted header with column labels for “#”, “Name”, and “Party”

loop through and print a string representation of each element in Democrat array list

// end for

// end displayQueueDem()

**displayQueueRepub()**

print “Queue 2: Republicans”

print formatted header with column labels for “#”, “Name”, and “Party”

loop through and print a string representation of each element in Republican array list

// end for

// end displayQueueRepub()

**displayQueueWhig()**

print “Queue 3: Whigs”

print formatted header with column labels for “#”, “Name”, and “Party”

loop through and print a string representation of each element in Whig array list

// end for

// end displayQueueWhig()

**displayAllQueues()**

call displayQueueDem()

call displayQueueRepub()

call displayQueueWhig()

// end displayAllQueues()

**displayBorder()**

print formatted decorative border

// end displayBorder()

**Stack**

**push(Presidents object)**

pre-increment *top* and add *object* to *stack*

// end push()

**pop()**

remove the element from *stack* at position *top* after decrementing *top* and return it

// end pop()

**peek()**

get element at position *top* from *stack* and return it

// end peek()

**isEmpty()**

return whether *top* is -1

// end isEmpty()

**isFull()**

return whether *top* is one less than the size of *stack*

// end isFull()

**displayStack(Stack stack)**

print formatted header with column labels for #, Name, and Party

print “TOP OF STACK”

while stack is not empty

peek at the top of *stack* and print result

pop *stack*

// end while

print “BOTTOM OF STACK”

// end displayStack()