COSC 3319 Data Structures Test 1 Spring 2017 Burris

You may not use a calculator on this test.

Answer five of the following seven questions **including question 2 and 3**. Number your questions on the answer sheets in ascending numeric order from 1 through 7 inclusive. Clearly write "Delete" on the answer sheets by the 2 questions you do not wish graded. Leave at least a one-inch margin at the top of every page. You may answer/delete multiple questions on the same page as long as you do so in consecutive ascending numeric order. Do not write on the back of pages. Staple your answer sheets (in ascending numeric order) on top of the test in the upper left hand corner. Write your name in the upper right hand corner of the answer sheets (first page). Turn the stapled bundle over and write your name in the upper right hand corner on the back of the test. A five point Road Map Fee (RMF) will be deducted for each instance of failure to follow instructions. You will not receive credit for material I cannot read or that is obstructed from my view by the staple. **Algorithms must be written using pseudo code emphasizing the notation (meta-language) taught in class. You may not use programming languages to express algorithms. The meta-language from class uses symbols such as =>, <=, and ← to express ideas.**

Warning: There are no "short" answers on this test. Tell me everything germane to the topic. Your performance is being compared to all other members of the class.

1) Marketing needs us to provide the ability to store 83 advertising records [AddrVerRec(J)] with logical addresses of 1 through 83 starting at logical address L_0 within a larger array containing non-related information for random access. Each advertising record will require 500 units of memory in the array. The current layout of memory follows:

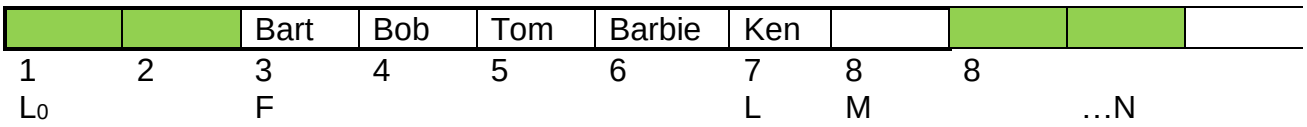
Other information	83 advertising records each 500 units in length	Additional information

0 L_{0-1} L_0 699986

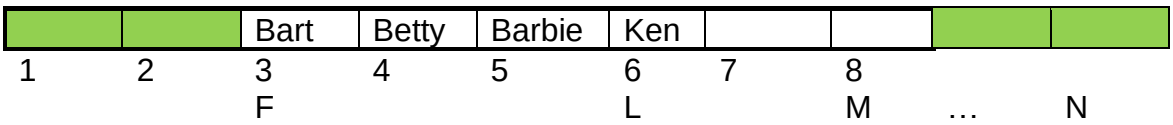
The exact starting address for our information/array will start at logical location L_0. Write the formula to calculate the address of any random advertising record.
Hint: Loc[AdderVerRec(J)] = ? for 1<= J <= 83.

2) **Write an algorithm to delete a person at a random location J from the following list without leaving a hole as demonstrated in class. The list may occupy locations for F <= J <=M. Items in locations greater than J must be moved down on deletion to maintain the list continuity. Remember to adjust the value of L on deletion.** The front of the list (base) is located at location "F" and location "F" will always contain the first item in the list unless the list is empty. The current last item in the list at location "L" and the last available location for insertion is "M" at location 8 in the example below. Other locations in the list are utilized for other purposes and may not be utilized by your algorithm. Management desires the boundary condition for the empty list to be L = F - 1. A sample for deleting Tom at location J = 5 follows.

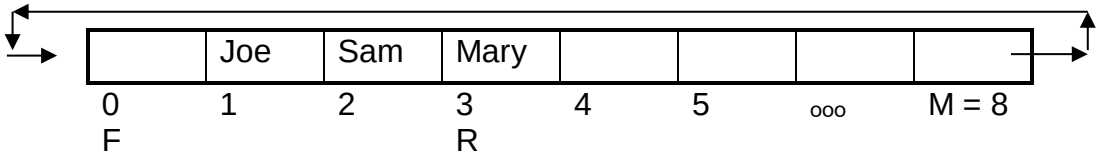
MyData: array(L_0 .. N) of item;
MyData before deletion of Tom at location J = 5, F = 3, and L = 7.



After deleting Tom at J = 5, F = 3 and L = 6.



3) We have been asked to **write an algorithm to insert items in a circular queue of M+1 items <u>allowing the use of all M+1 spaces</u>**. You must detect and report overflow. The queue below currently has 3 items and will store a maximum of 9 items, i.e., (M + 1) = 9. Hint: Add one to a counter on every insertion and subtract one on every deletion. Clearly state any initial conditions.



4) Developers may allocate space for data structures in the heap or stack. Compare the advantages and disadvantages of heap and stack memory. Consider both compiled and interpretive languages.

5) Consider the problem of maintaining multiple variable size list in a fixed amount of space. When overflow occurs we adjusted the space allocated to each stack in the list using two methods. State the two methods clearly explaining why each method was considered and used.

6) Consider the algorithm to maintain multiple stacks in a fixed region of memory with repacking. **First**, how do we know when every last unit of memory has been used? Your explanation must be very explicit. **Second**, why do we normally try to allocate more memory than will be required for the stacks. For maximum credit, approximate the overhead for repacking.

7) When possible, we allocated space to stacks initially based on their anticipated sizes. If no information on stack sizes was available, space was allocated equally. **First** develop the equal allocation formula. **Second**, what special consideration should be associated the space for the first stack if possible.

8) Dr. McGuire has assigned a lab in your favorite language, assembly, requiring one dimensional data structures (arrays, vectors). Each entry in the vector consists of 20 characters (C = 20) starting at memory address L0 = 17897 with logical subscripts from -10 through +20 inclusive. **Write a formula to compute the address of any random entry in the vector (array) evaluating all constants**. You must neatly show your work to receive credit.

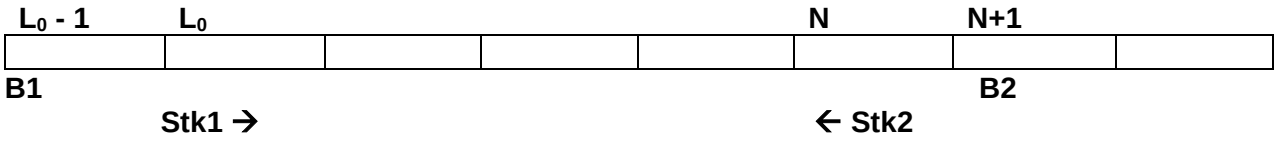
Hint: Loc[A[J]] ← L0 + 20 * (J + 10)

9) Assume a developer must determine which of sequentially allocated (contiguous) memory or dynamically allocated memory (linked list) would be best for implementing a data structure. What features (operations) would best lend their selves to dynamically allocated memory. **Number your responses**.

Hint: Directly from notes.

10) Developers may allocate space for data structures in the heap or stack. Compare and contrast the advantages and disadvantages of heap and stack memory. Consider both compiled and interpretive languages.

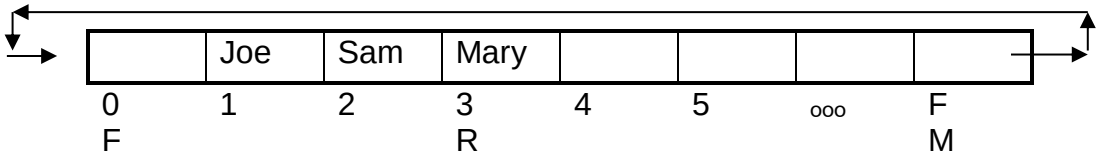
11) You have been tasked with writing software implementing two stacks subject to the following conditions. Each stack may occupy M units of memory. You only have M units of memory currently available hence both stacks must share the same space. Fortunately, we are reasonably sure the size of stack 1 plus the size of stack 2 will always be less than or equal to M. **Write the algorithms to insert and delete from both stacks checking for underflow and overflow.** You have been assigned the portion of memory in the diagram below from L_0 to N (use memory locations L_0 through N to store items in the stacks). Your algorithms must be general allowing any values for L_0 and N as long as L_0 is less than N. The bases for stack 1 and stack 2 appear in the diagram below including their direction of growth. **Clearly state the boundary condition for each stack to be empty**.



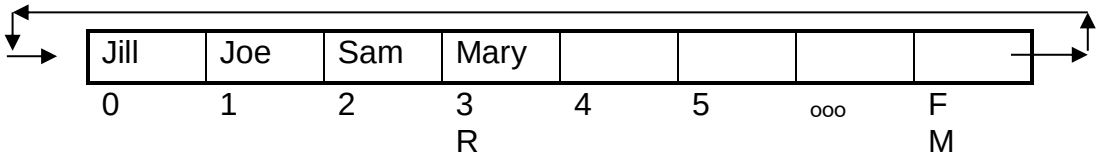
Hint: Directly from notes.

12) **Write a general algorithm to insert a new item at the front of the circular deque. Your algorithm must detect and report overflow as efficiently as possible.** As a sample, the deque would appear as follows before and after inserting Jill at the front. The deque is considered empty anytime F = R.

Before:



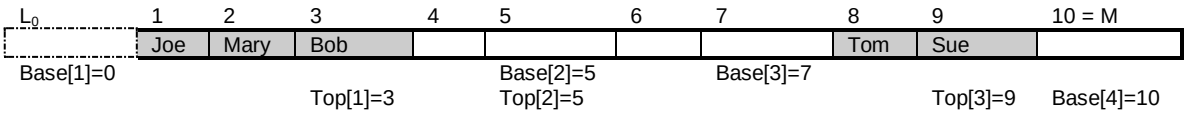
After:



Hint: Directly from notes.

13) **First**, write an algorithm to provide as closely as possible equal space for N stacks to be stored in the space from L0 through M in the diagram below. Be sure to clearly state any boundary conditions. **Second**, at what location should the front and rear pointers to the first queue be placed during storage allocation to insure the most efficient operation possible using the algorithms developed in class for repacking if a stack overflows?

The diagram below shows 3 queues. Space has not been allocated equally.



When queues overflow, memory is repacked to allow continued operation. Do not consider the repacking operation.

Hint: Directly from notes. Do the calculations.

14) We desire to store **multiple <u>queues</u>** (not stacks) using physical locations Min through Max. Queues grow from left to right only in the diagram (non-circular implementation). A queue is considered empty anytime F[J] = R[J]. Overflow occurs anytime R[J] > F[J+1]. For convenience, given N queues, we will utilize N + 1 front pointers with F[N + 1] = Max to detect overflow out of the last queue. As an initial condition, we always set F[1] = R[1] = Min – 1 for the first queue to maximize memory utilization.

Write the algorithms to insert and remove items from <u>queue J</u> subject to the above description detecting overflow and underflow. The diagram below shows 3 queues. Queue 1 contains 1 item, queue 2 is empty and queue 3 contains two items.

QSpace

	Joe			ooo		Bob	Tom	
Min-1	Min	Min+1	Min+2	ooo	Max-2	Max-1	Max	Max+1
F_1			F_2		F_3		F_4	
	R_1		R_2				R_3	

When queues overflow, memory is repacked to allow continued operation. Do not consider the repacking operation.

Insert:

```
If R[J] = B[J+1]
        Overflow
Else
        R[J] ← R[J] + 1
        QSpace[ R[J] ] ← y
```

Delete:

```
If F[J] = R[J] then
        Underflow
Else
        F[J] ← F[J+1]
        Y ← QSpace[ F[J] ]
End if
```

15) We desire to store **multiple <u>queues</u>** (not stacks) using physical locations Min through Max. Queues grow from left to right only in the diagram (non-circular implementation). A queue is considered empty anytime F[J] = R[J]. Overflow occurs anytime R[J] > F[J+1]. For convenience, given N queues, we will utilize N + 1 front pointers with F[N + 1] = Max to detect overflow out of the last queue. As an initial condition, we always set F[1] = R[1] = Min – 1 for the first queue to maximize memory utilization.

Write the algorithms to insert and remove items from <u>queue J</u> subject to the above description detecting overflow and underflow. The diagram below shows 3 queues. Queue 1 contains 1 item, queue 2 is empty and queue 3 contains two items.

QSpace

	Joe			ooo		Bob	Tom	
Min-1	Min	Min+1	Min+2	ooo	Max-2	Max-1	Max	Max+1
F_1			F_2		F_3		F_4	
	R_1		R_2				R_3	

When queues overflow, memory is repacked to allow continued operation. Do not consider the repacking operation.

Partial answer:
Delete:
If (F[J] = R[J])
    Underflow
Else
    F ← F + 1
    Y ← QSpace[F[J]]
EndIf

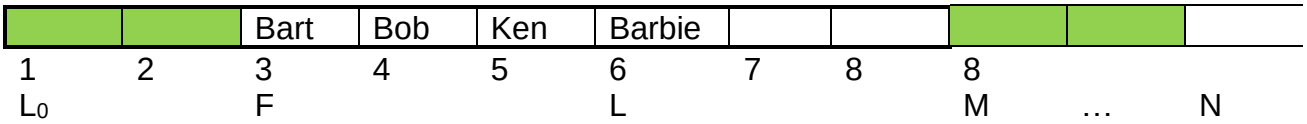
16) **<u>First</u>**<u>, write an algorithm to provide as closely as possible equal space for N queues (not stacks) as described in problem 1</u>. You must allow all queues to grow as much as possible prior to the first overflow! Be sure to clearly state any boundary conditions. **<u>Second</u>**<u>, at what location should the front and rear pointers to the first queue be placed during storage allocation to insure maximum use of the available space</u>?

Partial: F[J] = R[J] = floor[(J-1)/N * (Max – (Min-1)] + (Min-1) for J = 1..N

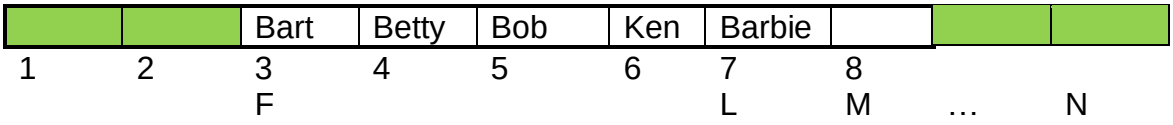
17) **Write an <u>algorithm</u> to insert a person at a random location J in the following list without destroying the current contents of the list for (F <= J < M). Items in locations J and greater than J must be moved up to prior to the insertion at J to maintain the list continuity.** The front of the list (base) is located at location “F” and <u>location “F” will always contain the first item in the list unless the list is empty</u>. The current last item in the list at location “L” and the last available location for insertion is “M” at location 8 in the example below. Other locations in the list are utilized for other purposes and may not be utilized by your algorithm. You must detect and report overflow.

Management desires the boundary condition for the empty list to be L = F - 1. Beware potential special cases for insertion and deletion! On insertion, be sure to adjust L to properly reflect the location of the last person in the list. A sample for inserting Betty at location J = 4 follows.

MyData: array(L_0 .. N) of item;
MyData before inserting Betty at location J = 4, L = 6.



After inserting Betty at J = 4, L = 7.



Partial answer:
Delete:
If (F[J] = R[J])
    Underflow
Else
    F ← F + 1
    Y ← QSpace[F[J]]
EndIf

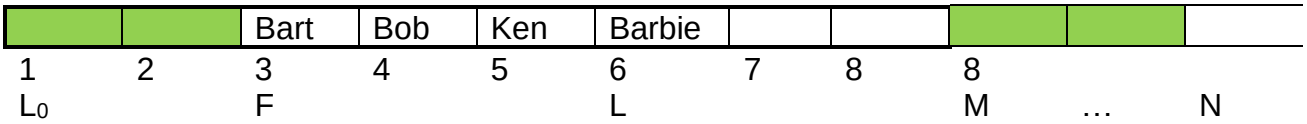
18) **First**, write an algorithm to provide as closely as possible equal space for N queues (not stacks) as described in problem 1. You must allow all queues to grow as much as possible prior to the first overflow! Be sure to clearly state any boundary conditions. **Second**, at what location should the front and rear pointers to the first queue be placed during storage allocation to insure maximum use of the available space?

Partial: F[J] = R[J] = floor[(J-1)/N * (Max – (Min-1)] + (Min-1) for J = 1..N

19) **Write an algorithm to insert a person at a random location J in the following list without destroying the current contents of the list for (F <= J < M). Items in locations J and greater than J must be moved up to prior to the insertion at J to maintain the list continuity.** The front of the list (base) is located at location "F" and location "F" will always contain the first item in the list unless the list is empty. The current last item in the list at location "L" and the last available location for insertion is "M" at location 8 in the example below. Other locations in the list are utilized for other purposes and may not be utilized by your algorithm. You must detect and report overflow.

Management desires the boundary condition for the empty list to be L = F - 1. Beware potential special cases for insertion and deletion! On insertion, be sure to adjust L to properly reflect the location of the last person in the list. A sample for inserting Betty at location J = 4 follows.

MyData: array(L_0 .. N) of item;
MyData before inserting Betty at location J = 4, L = 6.



After inserting Betty at J = 4, L = 7.

Partial:

```
If (L = M)
        Report overflow
Else
        For K = L downto J loop
                MyData[ K+1 ] ← MyData[ K ]
        End loop
        MyData[J] ← y
        L ← L + 1
```

20) Given the following declarations, **write a formula to locate any random record in the array evaluating all constants, i.e., calculate the actual physical memory address of element X[J]** relative to the start of the list using the indexing techniques developed in class. Assume the first unit of memory assigned by the translator is L_o. You need not do the actual arithmetic as long as you explicitly and neatly stipulate all values required for the calculation in your formulas.

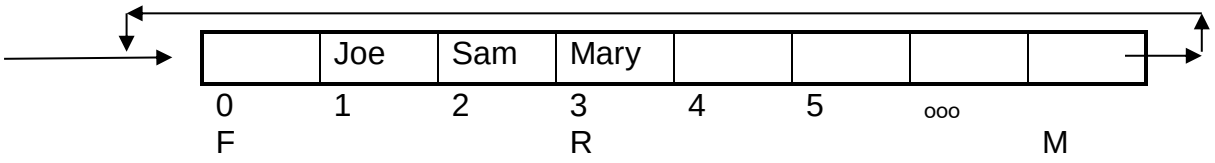
```
type MyType is record
        EmpID:  array(-6..6) of character;
        Name: array(1..10) of character;
        Age: integer;
        Sex: array(1..10) of character;
        Payrate: float;
end record;

X: array(-10..15) of MyType;
```

Assume a character occupies one unit of storage, an integer four units of storage, and a float consumes eight units of storage.

Hint: Loc[X[J]] ← base address + 45(J + 10), must show work for credit.

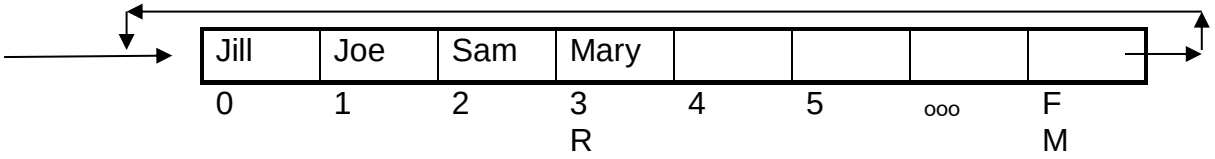
21) Assume a bounded linearly allocated circular deque of M+1 items as described below.

Bounded Linearly Allocated Circular Deque



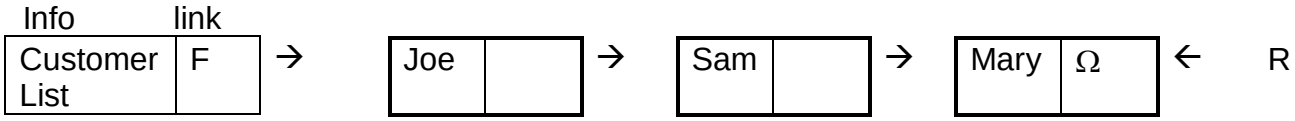
The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. Traditionally we have considered the deque as empty anytime F = R.

Write an algorithm to insert a new item at the front (F) of the deque (pointed to by F). Your algorithm must detect and report overflow as efficiently as possible. The above deque would appear as follows after inserting Jill at the front.



Hint: Directly out of notes.

22) In class we implemented a link allocated queue subject to the boundary conditions F := Ω and setting R := Location(F) when empty. **Write an algorithm to remove the last item in the queue effectively creating an input restricted deque (can remove from both ends but only insert on 1 end – the rear)**. Observe all boundary conditions including underflow and overflow. The sample queue/deque represents a customer list.



Partial:

```
If (F = Ω)
        Underflow
Else
        {locate node preceding last node}
        pt ← F
        While (pt.link <> Ω) loop
                pt ← pt.link
        End loop
        pt2 ← R
        Y ← pt2.info
        pt2 => avail
        {delete}
        If (pt = F)  -- deleting only node in list
                F ← Ω
                R ← Loc(F)
        Else       -- more than one item in list
                pt.link ← Ω
                R ← pt
        End if
End if
```

23) In class we developed a strategy to allow a list of dynamically allocated nodes to contain heterogeneous objects as opposed to homogeneous objects. Ada was used to illustrate the idea but the same approach can be used in other languages supporting inheritance including Java, C++, Ruby and Python. Explain the logic of the approach in detail. Explain its strengths and weaknesses. You may include some pseudo code as part of your discussion if desired. You must convenience me you have mastered the concept to receive full credit.

I have made my first pass to create the first test. It currently consists of 7 questions on 3 single spaced pages. It emphasizes sequentially allocated list (contiguous allocation) but does contain some material on dynamically allocated list and from object oriented programing (language neutral). I encourage you to look at the sample test for COSC 3319 on Blackboard, especially the first test for fall 2015. Start preparing now. Do not procrastinate. I will be checking to see if you are really giving me at least the 2 to 3 hours study time outside class per hour of class time required by the average student every class period.

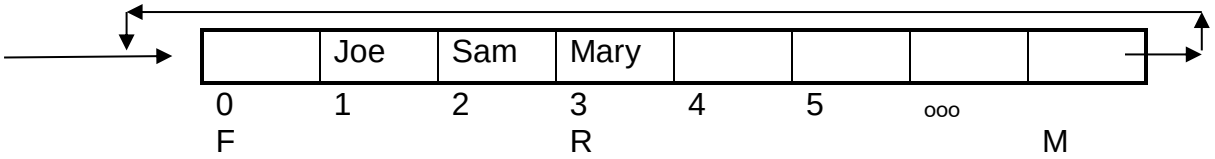
Our class meets at 8:A.M. so I would be happy to allow a 7:30 A.M. start if no one is inconvenienced. For convenience let us plan on the 7:30 start. If you cannot make the 7:30 start send me a private email or let me know at school so I can notify every one of the 8:00 A.M. start time.

Thank you,
Dr. B

The current instructions follow:

24) Assume a <u>bounded linearly allocated circular deque</u> of M+1 items as described below.

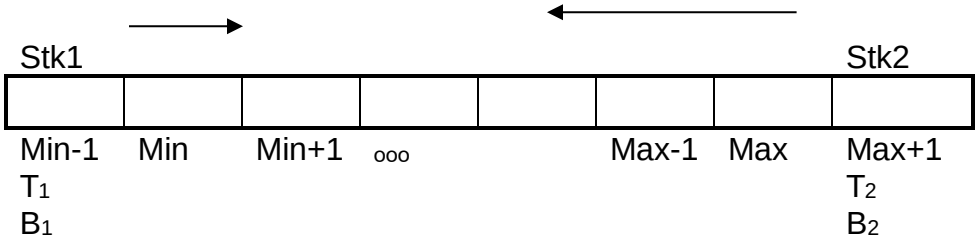
Bounded Linearly Allocated Circular Queue



The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. Traditionally we have considered the queue as empty anytime F = R.

Write an algorithm to remove the item at the rear of the deque (pointed to by R). Your algorithm must detect and report underflow as efficiently as possible.

25) Assume we need to store two stacks (Stk1 and Stk2) using physical locations Min through Max. The size of Stk1 plus the size of Stk2 should never exceed (Max-Min) + 1 units of storage though we must check for overflow on insertion and underflow on deletion. At any given time, we must allow for either stack to occupy the entire memory (locations Min through Max) if the other stack is empty. **Write the algorithms to insert (push) items in both Stk1 and Stk2 subject to the above conditions**. The stacks are considered empty any time $Top_I = Base_I$. The initial condition reflecting empty stacks is shown in the diagram below.



26) Assume the problem of maintaining N sequentially allocated stacks in a single contiguous (non-circular) region of memory as discussed in class. When a stack overflows we wish to move the existing stacks to new locations in the available space to allow continued execution with the goal of maximizing the time till the next potential overflow occurs. Explicitly state and explain the strategy or strategies you would use to accomplish this goal. Include any special strategy we might employ to minimize movement of a stack we anticipate to be the largest. You must explicitly state why you would employ this strategy with respect to the largest stack.

27) *Given the following declarations, **write a formula to locate any random record in the array evaluating all constants, i.e., calculate the actual physical memory address of element X[J].**. Assume the first unit of memory assigned by the translator is L_o. You need not do the actual arithmetic as long as you explicitly stipulate all values required for the calculation in your formulas.

```
type MyType is record
        EmpID:  array(-6..6) of character;
        Name: array(1..10) of character;
        Age: integer;
        Sex: array(1..10) of character;
        Payrate: float;
end record;

X: array(-20..19) of MyType;
```

Assume a character occupies one unit of storage, an integer two units of storage, and a float consumes four units of storage.

28) Assume we desire to store **multiple stacks** using physical locations Min through Max. Stacks grow from left to right only in the diagram (non-circular implementation). A stack is considered empty anytime B[J] = T[J] (the base of queue J is equal to the top of queue J). Overflow occurs anytime T[J] > B[J+1]. For convenience, given N queues, we will utilize N + 1 base pointers with B[N + 1] = Max to detect overflow out of the last stack. As an initial condition, we always set B[1] = T[1] = Min – 1 for the first queue to maximize memory utilization where Min is the first location that can actually hold data.

In class we developed an algorithm to provide equal initial space allocation for each of N stacks prior to application execution. Assume we wish to utilize 4 stacks. If the value of min is 20 and the value of Max is 100 calculate the initial base addresses allocating equal space to all 4 stacks using the formula developed in class. The sample diagram below shows 3 stacks. Stacks 1 and 2 are currently empty. Stack 3 is full and contains two items. **You must clearly show your work using the formula developed in class to receive credit!**

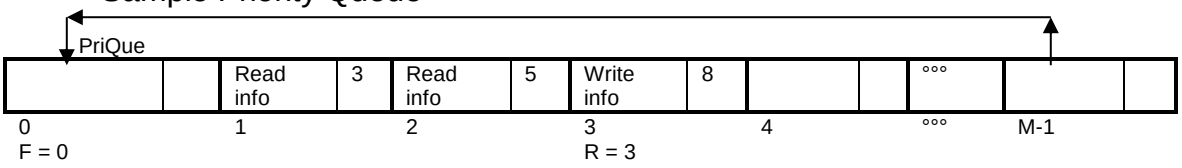
				ooo		Bob	Tom
Min-1	Min	Min+1	ooo		Max-2	Max-1	Max
B_1			B_2		B_3		B_4
F_1			F_2				F_3

29) [Do not jump to conclusions – read the entire question carefully] Our group has been assigned the task of placing disk request in a *Priority Queue* implemented in "M" contiguous (adjacent) memory locations occupying positions 0 through M-1. Priority queues are widely used in operating systems and commercial software. They are not queues in the traditional sense. All elements are removed from the front *but insertions generally do not occur at the rear*. The initial boundary condition is for F = R = 0. We may not assume the queue empties on a regular basis (all disk request are currently satisfied). We do not anticipate ever needing to track more than M - 1 transactions at a time. Since the list (priority queue) does not empty on a regular basis, we have been instructed to implement it as a circular list. The priority queue is empty any time F = R. Each entry in the queue contains information required to satisfy the disk read/write request and integer priority 1 to 10 with smaller numbers indicating a higher priority (1 is the highest).

Queue node format

Disk request information	priority

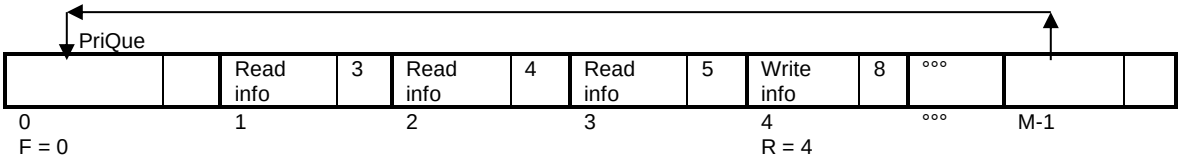
Sample Priority Queue



The deletion algorithm is:

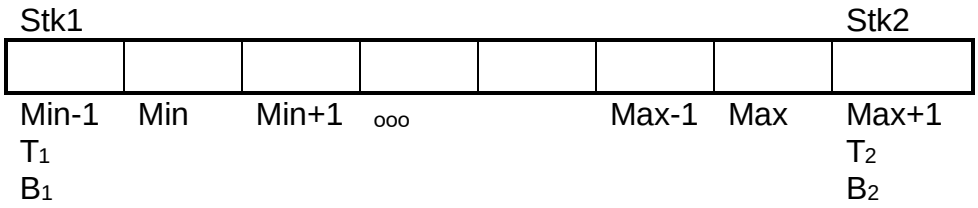
```
If (F = R) then
        Report underflow and terminate
Else
        F <- (F + 1) modulo M
        Y <- PriQue[ F ]
End if;
```

Your assignment is to write the algorithm to insert new disk request into the priority queue in *ascending order by priority* ensuring overflow does not occur. If two disk request will have the same (equal) priority, treat the existing request as the highest priority (the new request is lower priority). Assume a "read" with priority "4" is inserted in the above priority queue. After the insertion, the queue would appear as:



7) Assume the problem of maintaining N sequentially allocated stacks in a single contiguous (non-circular) region of memory as discussed in class. When a stack overflows we wish to move the existing stacks to new locations in the available space to allow continued execution with the goal of maximizing the time till the next potential overflow occurs. Explicitly state and explain the strategy or strategies you would use to accomplish this goal. Include any special strategy we might employ to minimize movement of a stack we anticipate to be the largest. You must explicitly state why you would employ this strategy with respect to the largest stack.

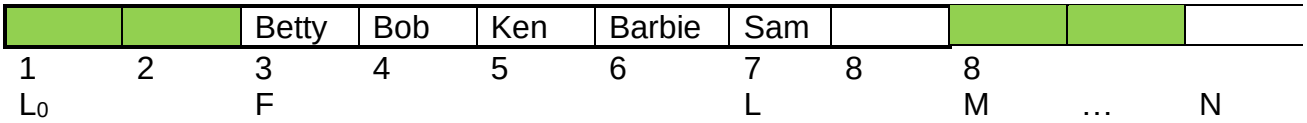
8) *Assume we need to store two stacks (Stk1 and Stk2) using physical locations Min through Max. The size of Stk1 plus the size of Stk2 should never exceed (Max-Min) + 1 units of storage though we must check for overflow on insertion and underflow on deletion. At any given time, we must allow for either stack to occupy the entire memory (locations Min through Max) if the other stack is empty. **Write the algorithms to insert (push) items in Stk1 and delete items from Stk2 subject to the above conditions**. The stacks are considered empty any time $Top_I = Base_I$. The initial condition reflecting empty stacks is shown in the diagram below.



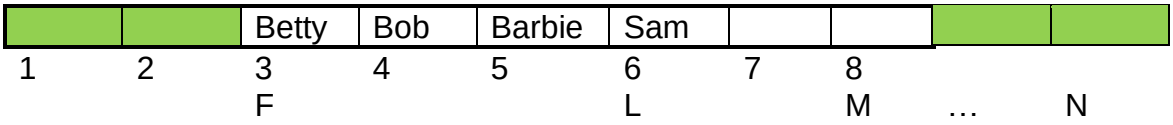
30) **Write an <u>algorithm</u> to delete a person at location J from the following list retaining lexicographic (sorted) order. Items in locations greater than J must be moved down to retain list continuity for 1 <= J <= L and L <= M.** The front of the list (base) is located at location "F." The current last item in the list at location "L" and the last available location for insertion is "M" = 8. Other locations in the list are utilized for other purposes and may not be utilized by your algorithm. You must detect and report underflow. Management desires the boundary condition for the empty list to be L = F – 1 (or L < F). On deletion, be sure to adjust L to properly reflect the location of the last person in the list. A sample for deleting Ken at location J = 5 follows.

MyData: array(L_0 .. N) of item;

MyData before deleting Ken at location J = 5, L = 7.



After deleting Ken L = 6.

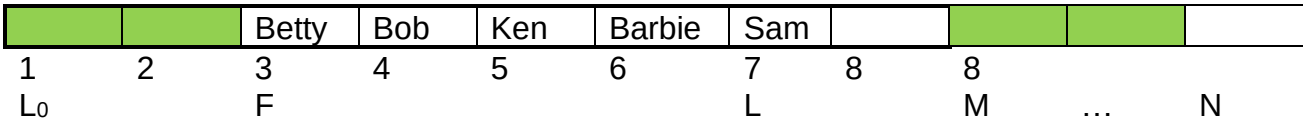


31) **Write an <u>algorithm</u> to insert a person at location J from the following list retaining lexicographic (sorted) order. Items in locations greater than J must be moved down to retain list continuity for 1 <= J <= L and L <= M.** The front of the list (base) is located at location "F." The current last item in the list at location "L" and the last available location for insertion is "M" = 8. Other locations in the list are utilized for other purposes

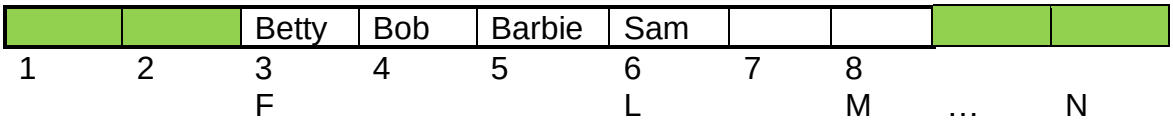
and may not be utilized by your algorithm. You must detect and report underflow. Management desires the boundary condition for the empty list to be L = F – 1 (or L < F). On deletion, be sure to adjust L to properly reflect the location of the last person in the list. A sample for deleting Ken at location J = 5 follows.

MyData: array(L_0 .. N) of item;

MyData before deleting Ken at location J = 5, L = 7.



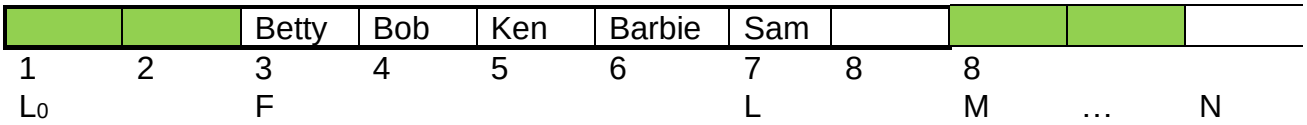
After deleting Ken L = 6.



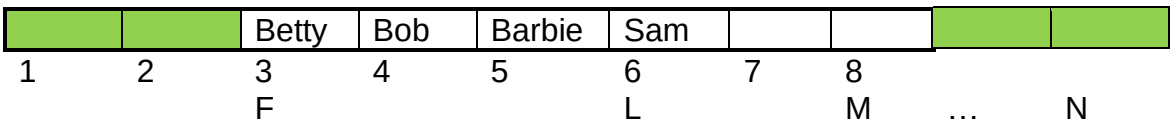
32) **Write an <u>algorithm</u> to insert a person at location J from the following list retaining lexicographic (sorted) order. Items in locations greater than J must be moved down to retain list continuity for 1 <= J <= L and L <= M.** The front of the list (base) is located at location "F." The current last item in the list at location "L" and the last available location for insertion is "M" = 8. Other locations in the list are utilized for other purposes and may not be utilized by your algorithm. You must detect and report underflow. Management desires the boundary condition for the empty list to be L = F – 1 (or L < F). On deletion, be sure to adjust L to properly reflect the location of the last person in the list. A sample for deleting Ken at location J = 5 follows.

MyData: array(L_0 .. N) of item;

MyData before deleting Ken at location J = 5, L = 7.



After deleting Ken L = 6.

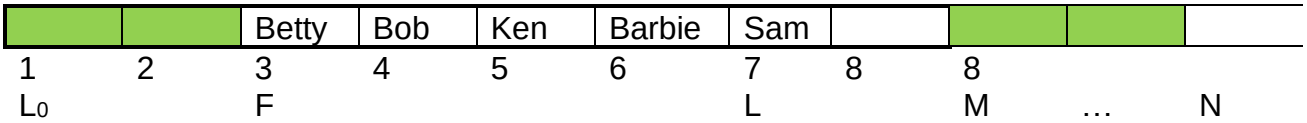


33) **Write an <u>algorithm</u> to insert a person at location J from the following list retaining lexicographic (sorted) order. Items in locations greater than J must be moved down to retain list continuity for 1 <= J <= L and L <= M.** The front of the list (base) is located at location "F." The current last item in the list at location "L" and the last available location for insertion is "M" = 8. Other locations in the list are utilized for other purposes

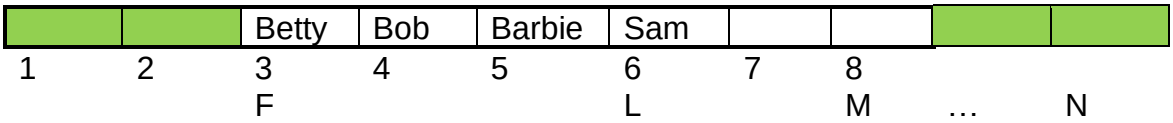
and may not be utilized by your algorithm. You must detect and report underflow. Management desires the boundary condition for the empty list to be L = F – 1 (or L < F). On deletion, be sure to adjust L to properly reflect the location of the last person in the list. A sample for deleting Ken at location J = 5 follows.

MyData: array(L_0 .. N) of item;

MyData before deleting Ken at location J = 5, L = 7.



After deleting Ken L = 6.



9) Given the following declarations, **write a formula to locate any random record** (e.g., base + offset) **in the array evaluating all constants, i.e., calculate the actual physical memory address of the location of X(J) in memory for any -30 <= J <= 20**. Assume the first unit of memory assigned by the translator is at location L_0 (base address). You need not do the actual arithmetic as long as you explicitly show/state all values/numbers required for the calculation and the formulas constituting your answer.

```
type MyType is record
        xAxis: array(-5..4) of character;
        yAxis: array(-5..4) of character;
        temperature: integer;
        airSpeed: character;
        chillFactor: float;
        locationt: Arrary(1..10) of character;
        label: Array(0..9) of character;
end record;

X: array(-30 .. 20) of MyType;
```

Assume a character occupies one unit of storage, an integer two units of storage, and a float consumes four units of storage. **You must show your work to receive credit!**

10) In class we developed the ability to store N stacks in a single contiguous section of memory from L_0 through M. Our new customer contract requires we use queues rather than stacks. Please write the algorithms to insert or remove an item from queue K. You must report underflow if the queue is empty. The front of all queues are maintained in the array front[1..N] and the rears of the queues are maintained in the array rear[1..N]. Clearly state the condition for overflow and underflow when using queues. Do not consider the repacking algorithm when overflow occurs. The storage is not to be treated as circular!

I have made the first pass at creating Test 1 in Data Structures. The first test covers all material from the start of the semester through page 27 (bounded linear allocated list) of the data structures notes. There is no coding on the test but you are expected to have mastered the concepts discussed in the coding examples. Traditionally those who have completed the "A" option on one or more labs are more successful on exams than those who have not completed the labs. The ability to read Ada is required. Anticipated test dates for the entire semester are listed in the syllabus. The test is three pages in length. I prefer questions requiring you to write algorithms, discuss a concept in great depth, or derive quantitative formulas. Sample test are available on Blackboard.

Our class meets from 8-9:30. If desired, I would be happy to allow everyone to start the test at 7:30 A.M. to relieve time pressure. This can only be done if it does not disadvantage any member of the class. If you cannot or do not wish to start at 7:30 A.M. please send me a private email so I can announce the start time as 8:00. That way no one will know who was unable to allow the early start time. I would appreciate being reminded in class to let everyone know the start time. Finally, if the university allows you to complete test with extra time at the testing center you must notify me of your desire to take advantage of this service at least two class periods in advance of the test. Test 1 contains more review material from previous classes than remaining test in the semester. Hence most students find it to be the easiest test of the semester. Please do not let the amount of review material to cause you to take the test for granted. The current instructions for the test follow:

COSC 3319 Data Structures Test 1 Fall 2013 Burris

You may not use a calculator on this test

Answer three of the following five questions **including question 1**. Number your questions on the answer sheets in ascending numeric order from one through five inclusive. Clearly write "Delete" on the answer sheets by the two questions you do not wish graded. Leave at least a one-inch margin at the top of every page. You may answer/delete multiple questions on the same page as long as you do so in consecutive ascending numeric order. Do not write on the back of pages. Staple your answer sheets (in ascending numeric order) on top of the test in the upper left hand corner. Write your name in the upper right hand corner of the answer sheets (first page). Turn the stapled bundle over and write your name in the upper right hand corner on the back of the test. A five point Road Map Fee (RMF) will be deducted for each instance of failure to follow instructions. You will not receive credit for material I cannot read or that is obstructed from my view by the staple.

Warning: There are no "short" answers on this test. Tell me everything germane to the topic. Your performance is being compared to all other members of the class.

11) Assume the following declarations in a program specification or body:

```
with Unchecked_Deallocation;

type person;
type personPt is access person;
type person is record                    -- Assume 12 units of storage.
        name: string(1..4);
        age: integer;
        next: personPt;
end record;

type car;
type carPt is access car;
type car is record              -- Assume 38 units of storage.
        manufacturer: string(1..30);  -- Ford, Mazda, etcetera
        modelYear: integer;
        next: personPt;
end record;

procedure Free is new Unchecked_Deallocation( person, personPt);
procedure Free is new Unchecked_Deallocation( car, carPt);

pt1, pt2: personPt;
carpt1, carpt2: carPt;

pt1 := new person'("Joe", 18, null);
Free(pt1);
carpt1 = new car("Ford", 2013, null);
Free(carpt1);
```

Compiled real time languages like Ada and "C" typically handle dynamic storage allocation very differently from interpretive languages like Java, PHP and Python. Compare and contrast "garbage collection" in Ada and Java. Your discussion should include who is responsible for garbage collection in each language? When does garbage collection occur? How many available storage lists does Ada create in the above code and why with respect to time/space tradeoffs. How does the Ada strategy compare to Java's approach? Your answer should be as inclusive as possible for both languages.

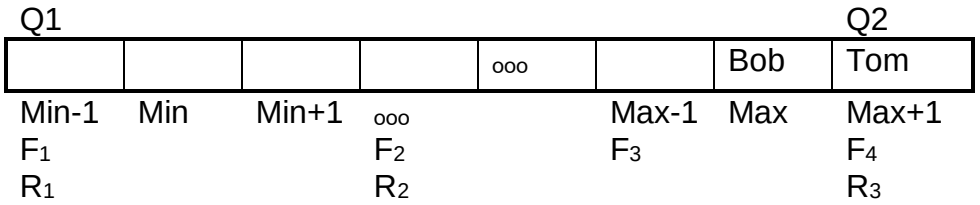
12) Assume we desire to store **multiple stacks** using physical locations Min through Max. Stacks grow from left to right only in the diagram (non-circular implementation). A stack is considered empty anytime B[J] = T[J] (the base of queue J is equal to the top of queue J). Overflow occurs anytime T[J] > B[J+1]. For convenience, given N queues, we will utilize N + 1 base pointers with B[N + 1] = Max to detect overflow out of the last stack. As an initial condition, we always set B[1] = T[1] = Min – 1 for the first queue to maximize memory utilization where Min is the first location that can actually hold data.

In class we developed an algorithm to provide equal initial space allocation for each of N stacks prior to application execution. Assume we wish to utilize 3 stacks. If the value of min is 22 and the value of Max is 104 calculate the initial base addresses allocating equal space to all 3 stacks using the formula developed in class. The sample diagram below shows 3 stacks. Stacks 1 and 2 are currently empty. Stack 3 is full and contains two items. **You must clearly show your work to receive credit!**

				ooo		Bob	Tom
Min-1	Min	Min+1	ooo		Max-2	Max-1	Max
B_1			B_2		B_3		B_4
F_1			F_2				F_3

13) Assume we desire to store N **queues** using physical locations Min through Max. Queues grow from left to right only in the diagram (non-circular implementation). A queue is considered empty anytime F[J] = R[J]. Overflow occurs anytime R[J] > F[J+1]. For convenience, given N queues, we will utilize N + 1 front pointers with F[N + 1] = Max to detect overflow out of the last queue. As an initial condition, we always set F[1] = R[1] = Min – 1 for the first queue to maximize memory utilization. When queues overflow, memory is repacked to allow continued operation.

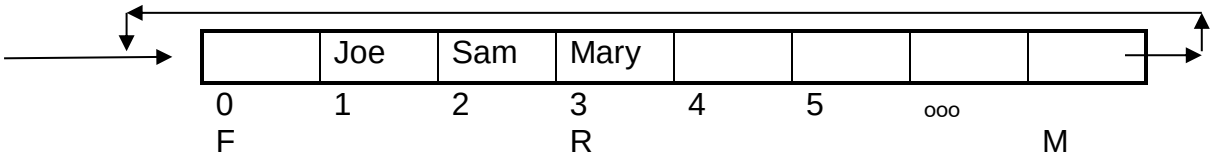
When overflow occurs 20% of memory is to be allocated equally between all N queues. The rest of memory is to be allocated based on dynamic growth. **First write an algorithm to calculate the available space for repacking when stack "K" overflows. Second calculate the value of α representing the amount of space each of the N stacks receives for equal allocation. Third calculate the value β representing the amount of space allocated for each unit of stack growth. Fourth, state any special considerations with respect to the new base locations for queue 1.**
A typical memory appears below for 3 queues:



Queues 1 and 2 are currently empty. Queue 3 contains two items.

14) Assume a <u>bounded linearly allocated circular queue</u> of M+1 items as described below.

Bounded Linearly Allocated Circular Queue



The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. Traditionally we have considered the queue as empty anytime F = R.

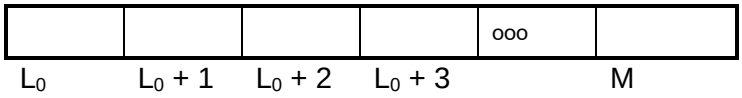
Our new customer contract requires the ability to store data in all M+1 locations (sacrifice run time to maximize storage utilization within the queue). Write the algorithms to insert and remove data from the circular queue based on the customer's specification. You must detect and inform the customer every time overflow or underflow occurs!

15) Given the following declarations, **write a formula to locate any random record** (e.g., base + offset) **in the array evaluating all constants, i.e., calculate the actual physical memory address of the location of X(J) in memory for any -10 <= J <= 200**. Assume the first unit of memory assigned by the translator is at location L_o (base address). You need not do the actual arithmetic as long as you explicitly show/state all values required for the calculation and their relationship. Do not be concerned with overflow.

```
type MyType is record
        Name: array(1..10) of character;
        Age: integer;
        Sex: character;
        Payrate: float;
        Department: Arrary(1..10) of character;
        PhoneNumber: Array(-12..12) of character;
        Title: Array(0..15) of character;
end record;

X: array(-10 .. 200) of MyType;
```

Assume a character occupy's one unit of storage, an integer two units of storage, and a float consumes four units of storage. **You must show your work to receive credit!**

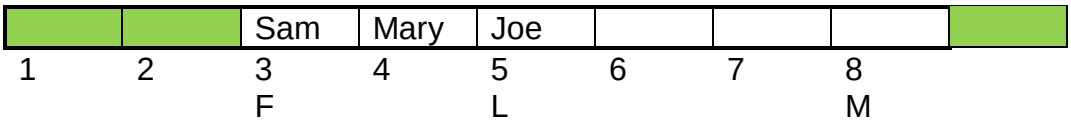
6) Assume the problem of maintaining N sequentially allocated stacks in a single contiguous (non-circular) region of memory as discussed in class. **State and describe at least two strategies discussed in class for initially allocating space for the stacks. You must describe the circumstances under which each strategy is the best**. **Please describe any special considerations in placing the stacks to minimize overhead if memory must be repacked. In general how much of the space can be utilized while still expecting adequate performance in terms of speed of response. As part of your answer, develop an algorithm to calculate Base[J] = Top[J] for 1 <= J <=N assuming equal space is to be provided initially for all N stacks using memory locations L_0 + 1 through M in the diagram below.**



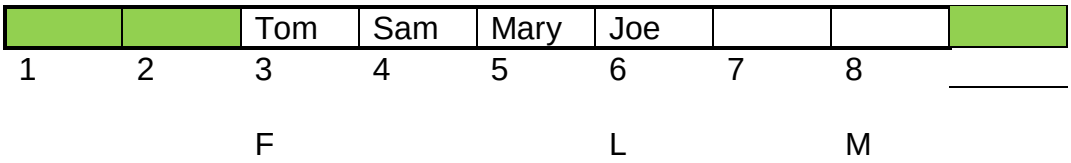
7) Write an algorithm to insert a new "item" at a random location "J," where F <= J <= L, in a sequentially allocated list occupying locations "F" through "M" without destroying existing items in the list. The current last element in the sample list below is at location "L."

MyData: array(L_0 .. M) of item;

MyData before insertion at location J = 3:



MyData after inserting Tom at random location J:



16) Given the following declarations, **write a formula to locate any random record** (e.g., base + offset) **in the array evaluating all constants, i.e., calculate the actual physical memory address of the location of X(J) in memory for any -30 <= J <= 20**. Assume the first unit of memory assigned by the translator is at location L_o (base address). You need not do the actual arithmetic as long as you explicitly show/state all values/numbers required for the calculation and the formulas constituting your answer.

```
type MyType is record
        xAxis: array(-5..4) of character;
        yAxis: array(-5..4) of character;
        temperature: integer;
        airSpeed: character;
        chillFactor: float;
        locationt: Arrary(1..10) of character;
        label: Array(0..9) of character;
end record;

X: array(-30 .. 20) of MyType;
```

Assume a character occupies one unit of storage, an integer two units of storage, and a float consumes four units of storage. **You must show your work to receive credit!**

17) Assume the problem of maintaining N sequentially allocated stacks in a single contiguous (non-circular) region of memory as discussed in class. The decision has been made to allocate space for N = 4 stacks equally in the available space using locations 75 through 1,404. The first item in stack 1 should physically occupy memory location 75. **First clearly exhibit the formula to do the calculations in symbolic form (no constants) to determine the initial stack base and top. Next exhibit the formula showing all values required to complete the calculation for each of the four stacks.** You need not physically do the calculations, just exhibit the formula an appropriate number of times with constants appropriate to the computation for each of the 4 stacks. Be sure to properly provide for detection of overflow in all stacks using the algorithms we developed in class.

If the words "symbolic" and "constant" bother you, consider a square with length = 56 feet and width = 42 feet. The area of the square in symbolic format is "area = length * width." The area using constants is "area = 56 * 42."

18) In class we implemented a link allocated queue subject to the boundary conditions F := Ω and setting R := Location(F) when empty. Write an algorithm to insert a new item at the front of the list (pointed to by F). Observe all boundary conditions including underflow and underflow. This will result in an output restricted dequeue.

3) Given the following declarations, **write a formula to locate any random record in the array evaluating all constants, i.e., calculate the actual physical memory address of the location Loc[X(J)] in memory for any -15 <= J <= 366**. Assume the first unit of memory assigned by the translator is L_o. You need not do the actual arithmetic as long as you explicitly stipulate all values required for the calculation.

```
type MyType is record
        Name: array(1..10) of character;
        Age: integer;
        Sex: character;
        Payrate: float;
end record;

X: array(-15..366) of MyType;
```

Assume a character occupies one unit of storage, an integer two units of storage, and a float consumes four units of storage.

19) Choice (answer only one of the following):

A) Assume the problem of maintaining N sequentially allocated stacks in a single (non-circular) region of memory as discussed in class. Discuss each of the following considerations in detail.
   1) How space should initially be allocated to each of the N stacks?
   2) How space should be redistributed when stack overflow occurs?
   3) How to minimize the amount of stack entries that must be moved during repacking?
   4) How to balance the trade off in runtime CPU utilization versus the desire to maximize the efficient use (all) of main memory?
   5) How do we know when the last unit of main memory has been exhausted?

B) In class we developed algorithms to store N stacks in a fixed amount of (contiguous) sequentially allocated memory from L_0 through M. When memory overflowed, we repacked the space when possible to continue operation. Rather than stacks, we now must utilize queues. *Write the algorithms to insert and delete items from the "Jth" queue. Your algorithms must detect overflow and underflow. Clearly state the boundary condition for overflow. We consider a queue empty anytime F[J] = R[J]. Be sure you do not have to treat the first or last queue as a special case in your algorithm*. Memory must not be treated as "circular." Do not address the problem of what should be done when overflow occurs.

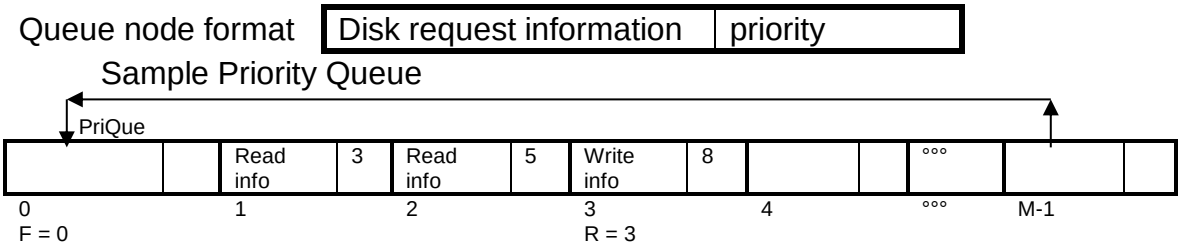
5) In class we implemented a link allocated queue subject to the boundary conditions F := Ω and setting R := Location(F) when empty. Write an algorithm to remove the last item in the list (pointed to by R). Observe all boundary conditions including underflow. This will result in a input restricted dequeue.

{one node in list}

Hint:

```
If F = omega
   Underflow
Else
   If F = R {one node in the list}
      Y <= infor[R]
      R => Avail
      R <-- Location(F)
   Else  {multiple nodes, find node preceding R}
         pt <-- F
         While (pt.next != R) loop
                  pt <-- pt.next
         End loop
         Y <-- R.info
         R => Avail
         R <-- pt
         R.next <-- omega
   End if
End if
```

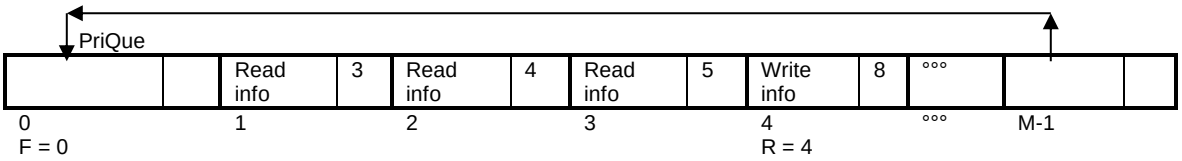
5) [Do not jump to conclusions – read the entire question carefully] Our group has been assigned the task of placing disk request in a *Priority Queue* implemented in "M" contiguous (adjacent) memory locations occupying positions 0 through M-1. Priority queues are widely used in operating systems and commercial software. They are not queues in the traditional sense. All elements are removed from the front ***but insertions generally do not occur at the rear***. The initial boundary condition is for F = R = 0. We may not assume the queue empties on a regular basis (all disk request are currently satisfied). We do not anticipate ever needing to track more than M - 1 transactions at a time. Since the list (priority queue) does not empty on a regular basis, we have been instructed to implement it as a circular list. The priority queue is empty any time F = R. Each entry in the queue contains information required to satisfy the disk read/write request and integer priority 1 to 10 with smaller numbers indicating a higher priority (1 is the highest).



The deletion algorithm is:

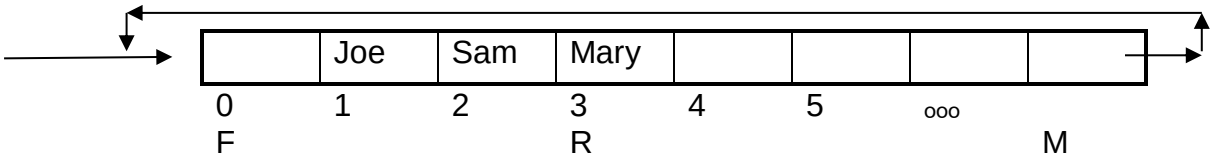
```
If (F = R) then
        Report underflow and terminate
Else
        F <- (F + 1) modulo M
        Y <- PriQue[ F ]
End if;
```

Your assignment is to write the algorithm to insert new disk request into the priority queue in *ascending order by priority* ensuring overflow does not occur. If two disk request will have the same (equal) priority, treat the existing request as the highest priority (the new request is lower priority). Assume a "read" with priority "4" is inserted in the above priority queue. After the insertion, the queue would appear as:



20) Assume a <u>bounded linearly allocated circular queue</u> of M+1 items as described below.

Bounded Linearly Allocated Circular Queue

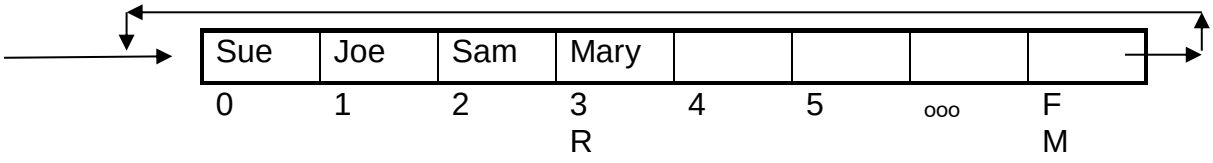


The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. Traditionally we have considered the queue as empty anytime F = R.

Our new customer contract requires the ability to insert data as the new first item in the queue (at the front) in addition to the normal queue operations creating an output restricted deque. Write the algorithm to insert new data at the front of the circular queue. You must detect and inform the customer every time overflow occurs meeting the criteria the output restricted deque is empty any time F = R.

As an example, the above structure would appear as shown below after inserting Sue at the front.

Bounded Linearly Allocated Circular Queue



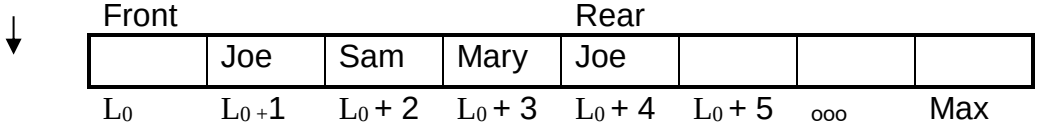
21) In class we developed an algorithm to sort a large number of records into related groups (secret agents by job classification) in a single pass over the data. Given the following information, write the sorting algorithm. I am not looking for code but will take it. I prefer a general algorithm.

```
type jobClassification is (banking, photography, reconnaissance, dirtyTricks,
        interrogation, politician, lover);

type secretAgent;
type agentPoint is access secretAgent;
type secretAgent is record
        job:  jobClassification;
        agentName: string(1..40);
        currentLocation: string(1..80);
        next: agentPoint;
end record;

sortArray: array(1..20,000) of secretAgent;
```

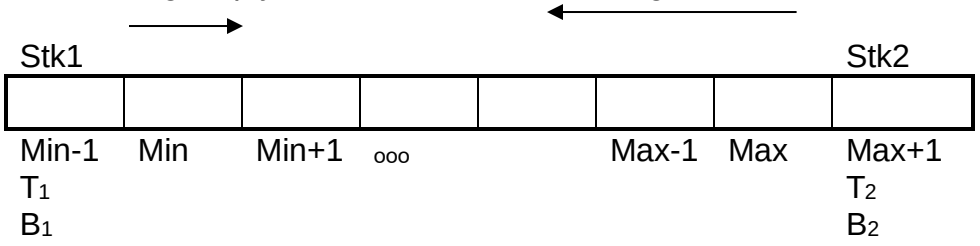
22) Memory for list structures may be allocated contiguously (arrays) or dynamically (linked). In class we discussed 9 operations on lists and discussed the memory allocation scheme under which they were most efficient, e.g., a random insertion immediately prior to the k^{th} node. List at least 7 of these operations on list including random insertion numbering your responses. For each operation, you must state which memory allocation scheme tends to support the operation most efficiently and briefly explain why.

23) Assume we implement the queue as an array of Max items. The variable "front" points to the next available element in the queue. "Rear" points to the last item in the queue. We assume all insertions at the rear and all removals from the front. Front = rear will be the boundary condition to indicate an empty queue. For initialization, we set front = rear = L_0. We implement the queue under the assumption that the queue empties on a regular basis, i.e., the front = rear when the last item is removed from the queue. When the queue becomes empty, we reset front = rear = L_0 to reclaim previously used space (prevent the queue from constantly growing in the same direction). **Write the insertion and deletion algorithms!**

Sample queue:

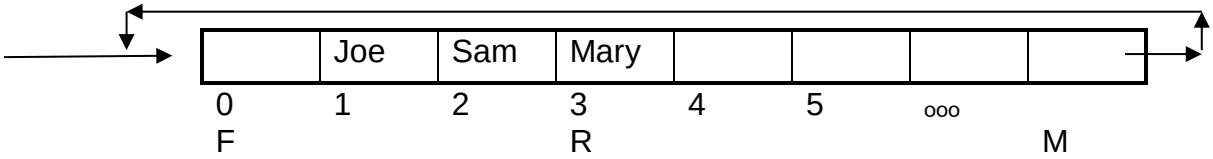


24) Assume we need to store two stacks (Stk1 and Stk2) using physical locations Min through Max. The size of Stk1 plus the size of Stk2 should never exceed (Max-Min) + 1 units of storage though we must check for overflow on insertion and underflow on deletion. At any given time, we must allow for either stack to occupy the entire memory (locations Min through Max) if the other stack is empty. **Write the algorithms to insert (push) items in both Stk1 and Stk2 subject to the above conditions**. The stacks are considered empty any time $Top_I = Base_I$. The initial condition reflecting empty stacks is shown in the diagram below.



25) Assume a bounded linearly allocated circular queue of M items as described below.

Bounded Linearly Allocated Circular Queue

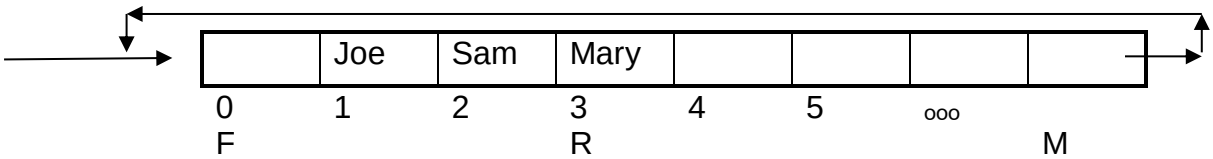


The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. We wish to treat the queue as empty anytime F = R.

Our new customer contract requires the ability to treat the above data structure as an input restricted deque (can only insert at the rear but can delete on both ends). Write the algorithm to insert data at the front of the deque observing all boundary conditions. In the above diagram, Tom would be located at position 0 if inserted at the front of the deque.

26) Assume a bounded linearly allocated circular queue of M items as described below.

Bounded Linearly Allocated Circular Queue



The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. We wish to treat the queue as empty anytime F = R.

Our new customer contract requires the ability to treat the above data structure as an input restricted deque (can only insert at the rear but can delete on both ends). Write the

algorithm to insert data at the front of the deque observing all boundary conditions. In the above diagram, Tom would be located at position 0 if inserted at the front of the deque.

2) In general, data structures may be allocated statically in contiguous memory or dynamically. The choice is typically based on the operations to be performed on the data structure. Examples include random insertion, random deletion, accessing the "n" item, and breaking a list into 2 or more list. Based on the discussion in class, list as many operations as possible favoring contiguous memory allocation with a brief explanation. NUMBER YOUR RESPONSES! Now repeat these steps for operations favoring dynamic allocation.

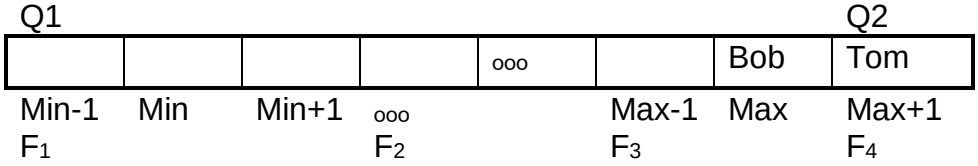
I have made the first pass to create Test 1. My goal for you is to be able to replace me as the instructor next semester. Please do not put off studying till the last minute. Bring a watch and allow 10 minutes per question with 5 to 10 minutes to review the questions prior to answering them. Some questions will be easier than other. Some will require more time.

You need to have mastered basic material prior to the test. There will not be sufficient time to "figure it out" during the test. I recommend reviewing test from previous semester on Blackboard. We cover the same material every semester and tests tend to have extensive overlap. The purpose of the question you "must answer" is to encourage you to not neglect topics you think I will not ask. Test dates are listed in the syllabus on Blackboard.

The test covers sequentially allocated list through page 26 (inserting a new record in sorted order). It does not cover dynamically allocated lists. I have placed my most recent version of the notes "Data Structures" on Blackboard to make sure we are looking at the same content. The instructions for the test follow:

27) Assume we desire to store **multiple queues** using physical locations Min through Max. Queues grow from left to right only in the diagram (non-circular implementation). A queue is considered empty anytime F[J] = R[J]. Overflow occurs anytime R[J] > F[J+1]. For convenience, given N queues, we will utilize N + 1 front pointers with F[N + 1] = Max to detect overflow out of the last queue. As an initial condition, we always set F[1] = R[1] = Min – 1 for the first queue to maximize memory utilization.

Write the algorithms to insert and remove items from queue J subject to the above description detecting overflow and underflow. The diagram below shows 3 queues. Queues 1 and 2 are currently empty. Queue 3 contains two items.

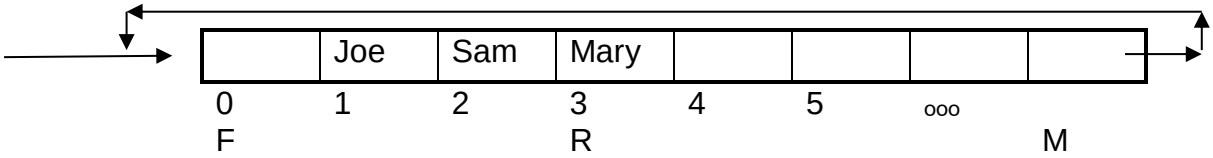


R_1 R_2 R_3

When queues overflow, memory is repacked to allow continued operation. Do not consider the repacking operation.

28) Assume a <u>bounded linearly allocated circular queue</u> of M+ 1 items as described below.

Bounded Linearly Allocated Circular Queue



The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. We wish to treat the queue as empty anytime F = R.

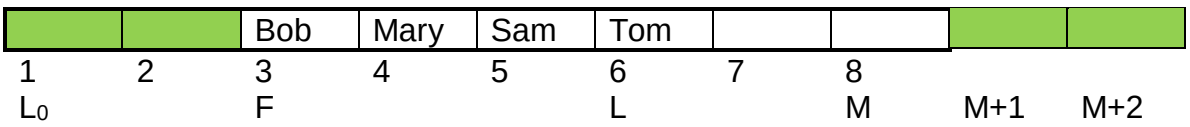
Our new customer contract requires the ability to treat the above data structure as an input restricted deque (can only inset at the rear but can remove on both ends). Write the algorithm to remove data from the front of the deque observing all boundary conditions. If Joe was removed from the front in the above diagram, Sam and Mary would occupy positions 2 and 3 respectively.

29) In class we implemented a link allocated queue subject to the boundary conditions F := Ω and setting R := Location(F) when empty. Write an algorithm to insert a new item at the front of the list (pointed to by F). Observe all boundary conditions including underflow and underflow. This will result in an output restricted dequeue.

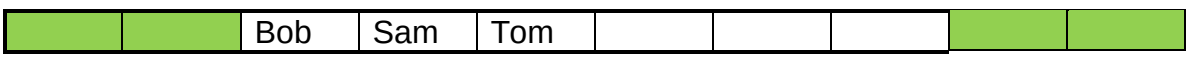
30) Write an algorithm to delete an "item" at a random location "J," where F <= J <= L, in a sequentially allocated list occupying locations "F" through "M" without destroying existing items in the list. The current last element in the sample list below is at location "L."

As an example, given MyData: array(L_0 .. M) of item;

MyData before deletion of Mary at location 4:



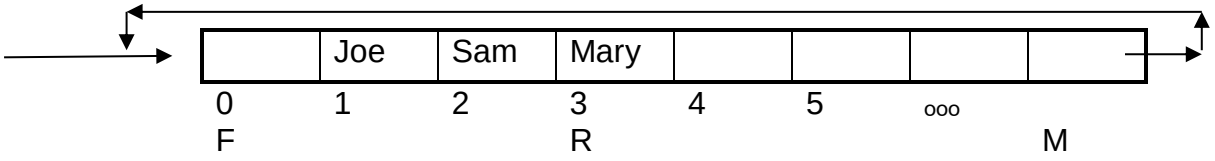
MyData after deleting Mary at location J = 4:



1	2	3	4	5	6	7	8
		F		L			M

31) Assume a <u>bounded linearly allocated circular queue</u> of M+1 items as described below.

Bounded Linearly Allocated Circular Queue

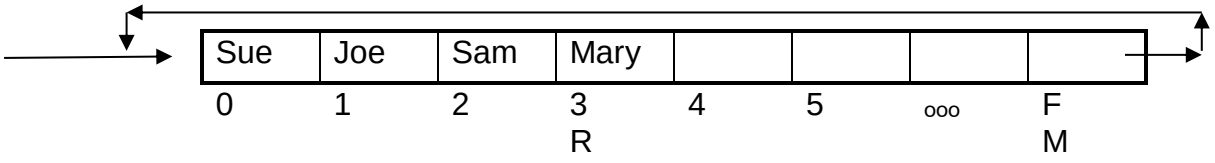


The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. Traditionally we have considered the queue as empty anytime F = R.

Our new customer contract requires the ability to insert data as the new first item in the queue (at the front) in addition to the normal queue operations creating an output restricted deque. Write the algorithm to insert new data at the front of the circular queue. You must detect and inform the customer every time overflow occurs meeting the criteria the output restricted deque is empty any time F = R.

As an example, the above structure would appear as shown below after inserting Sue at the front.

Bounded Linearly Allocated Circular Queue



32) Choice (answer only one of the following):

C) Assume the problem of maintaining N sequentially allocated stacks in a single (non-circular) region of memory as discussed in class. Discuss each of the following considerations in detail.

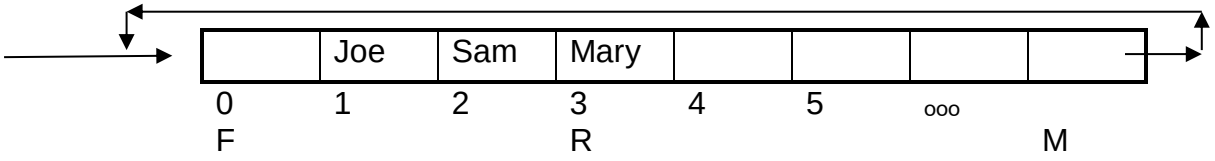
6) How space should initially be allocated to each of the N stacks?
7) How space should be redistributed when stack overflow occurs?
8) How to minimize the amount of stack entries that must be moved during repacking?
9) How to balance the trade off in runtime CPU utilization versus the desire to maximize the efficient use (all) of main memory?
10) How do we know when the last unit of main memory has been exhausted?

D) In class we developed algorithms to store N stacks in a fixed amount of (contiguous) sequentially allocated memory from L_0 through M (physically using locations L_0 + 1 through M). When memory overflowed, we repacked the space when possible to continue operation. Rather than stacks, we now must utilize queues. <u>*Write the*</u>

algorithms to insert and delete items from the "J[th]" queue. Your algorithms must detect overflow and underflow. Clearly state the boundary condition for overflow. We consider a queue empty anytime F[J] = R[J]. Be sure you do not have to treat the first or last queue as a special case in your algorithm. Memory must not be treated as "circular." Do not address the problem of what should be done when overflow occurs.

33) Assume a bounded linearly allocated circular queue of M+1 items as described below.

Bounded Linearly Allocated Circular Queue

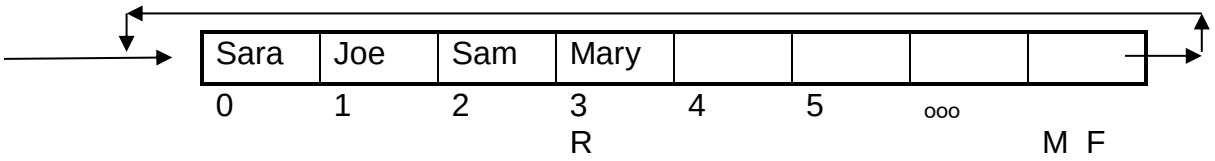


The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. Traditionally we have considered the queue as empty anytime F = R.

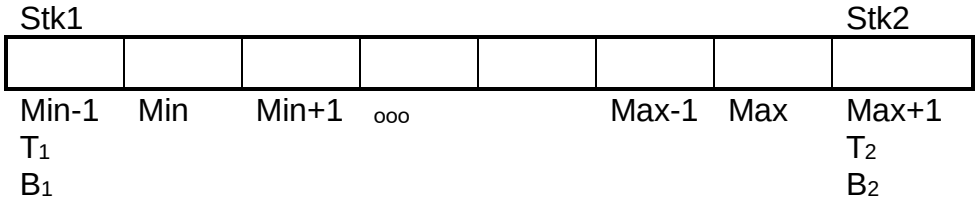
Our new customer contract requires the ability to insert at the front of the queue in addition to the normal queue operations creating an output restricted deque. Write the algorithm to insert new data at the front of the circular deque. You must detect and inform the customer every time overflow occurs meeting the criteria the output restricted deque is full any time F = R.

As an example, the above structure would appear as shown below after adding Sara at the front.

Bounded Linearly Allocated Circular Queue

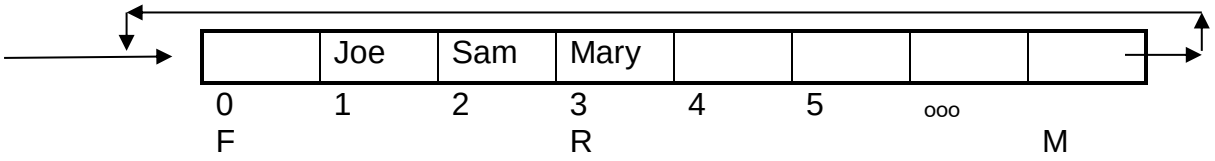


5) Assume we need to store two stacks (Stk1 and Stk2) using physical locations Min through Max. The size of Stk1 plus the size of Stk2 should never exceed (Max-Min) + 1 units of storage though we must check for overflow on insertion and underflow on deletion. At any given time, we must allow for either stack to occupy the entire memory (locations Min through Max) if the other stack is empty. Write the algorithms to insert (push) items in to both Stk1 and Stk2 subject to the above conditions. The stacks are considered empty any time $Top_I = Base_I$. The initial condition reflecting empty stacks is shown in the diagram below.



34) *Assume a <u>bounded linearly allocated circular deque</u> of M+1 items as described below.

Bounded Linearly Allocated Circular Queue



The above diagram reflects the contents after inserting three items. We assume the boundary condition that F <- R <- 0 initially. Traditionally we have considered the queue as empty anytime F = R.

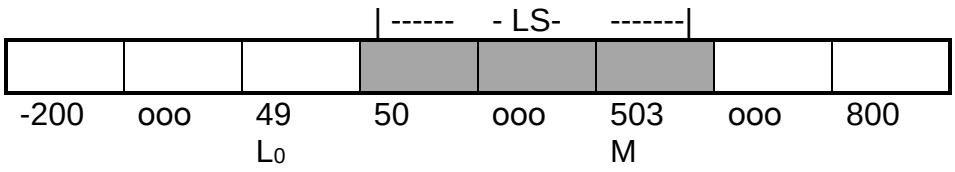
Write an algorithm to count the number of items in the list on demand as the value of a variable "knt."

3) For each of the following operations explain why static (contiguous) or linked (dynamic) memory allocation is usually best and why. You must submit you results in the order shown clearly labeled a through f.

a) randomly access the k^{th} node
b) randomly insert just before or after the k^{th} node
c) randomly delete the k^{th} node
d) combine two or more list into one list
e) split a list into two or more list
f) make a copy of a list

35) Assume the problem of maintaining N sequentially allocated stacks in a single contiguous (non-circular) region of memory as discussed in class. The decision has been made to allocate space for N = 4 stacks equally in the available space using locations 50 through 503 of the array "LS" allocated to support the space ship launch and navigation system. L_0 will be at location 49. "M"= 503 is the last usable element of the array for our purposes.

LS: Array[-200..800] of LaunchData; -- LS stands for launch space.



First develop and clearly exhibit the formula to do the calculations in symbolic form (no constants) to determine the initial stack bases and tops. Next exhibit the formula showing all values required to complete the calculation for each of the four stacks. You need not physically do the calculations, just exhibit the formula an appropriate number of times with constants appropriate to the computation for each of the 4 stacks. Be sure to properly provide for detection of overflow in all stacks using the algorithms we developed in class.

If the words "symbolic" and "constant" bother you, consider a square with length = 56 feet and width = 42 feet. The area of the square in symbolic format is "area = length * width." The area using constants is "area = 56 * 42."