**COSC 3319 Data Structures Test 2 Spring 2016 Burris**

Answer five questions **including questions 1**. Number the questions on the answer sheets in ascending numeric order from 1 through 9 inclusive. **You should be able to answer every question on the test!** Clearly write “Delete” on the answer sheets by the 4 questions you do not wish graded. Leave at least a one-inch margin at the top of every page. You may delete one or more questions on any page as long as the questions are in ascending 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.

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. From a professional standpoint, you should be able to answer every question. After the test, please review any material you feel needs more work.

1. Process the following precedence relations using the Topological Sort Algorithm taught in class. **You must do both parts “A” and “B” to receive credit**. To receive credit you must follow the algorithm without deviation.
2. **First process all relations in the order specified below to complete the initial data structure as indicated below using the algorithm taught in class.** Label your answer as “Part A” using the following format. **You must use all the relations in the indicated order.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | QLink | Betty | Bob | Joe | Kim | Mat | Sally | Sue | Tom |
| Count |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Top |  | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω |

Joe < Tom, Sally < Kim, Joe < Betty, Kim < Betty, Tom < Betty, Sue < Tom, Mat < Bob,

Sally < Sue, Bob < Betty

1. ***Second, make a copy of the data structure produced in Part A and label it “Part B*.” Utilize the count fields to build the linked allocated queue utilized by our algorithm. Complete the topological sort. The count fields of all tasks should be decremented appropriately and if any go to zero, they should be utilized as links to implement the queue** (all link pointers adjusted). If a loop is encountered (there is no solution), state which tasks form the loop.

***HINT: first 2 qlinks to Kim followed by Sue in the following sample.***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | QLink | Betty | Bob | Joe | Kim | Mat | Sally | Sue | Tom |
| Count | F = Kim | 1 | 1 | 2 | ~~0~~ Sue | 1 | 3 | 0 | 3 |
| Top |  |  |  |  | Ω |  | Ω |  |  |

Hint: final links

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | QLink | Betty | Bob | Joe | Kim | Mat | Sally | Sue | Tom |
| Count | F = Joe | 0 | Sue | Mat | ~~Tom~~ | Sally | Bob | Kim | Betty |
| Top |  |  |  |  | Ω |  | Ω |  |  |

Sort: Joe, Mat, Sally,Bob, Sue, Kim, Tom, Betty

1. Write an algorithm to insert a new node pointed to by Pt to the right of the node pointed to by IP. A sample list appears below. Your algorithm must cover every possible case. Assume the list is empty if Pt 🡨 Ω. In the sample, IP points to May and Pt to Jake. Jake will follow May once the insertion is complete.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pt1 | Tom. | 27 |  |  | Sam | 35 |  |  | May | 47 |  |  | Hope | 8 | Ω |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | IP |  | Pt |  |  | Jake | 10 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

The node format is:

|  |  |  |
| --- | --- | --- |
| Name | Age | Next |

Hint:

If (pt1 = Ω)

Pt.next 🡨 Ω

Pt1 🡨 pt

Else

Pt.next 🡨 IP.next

IP.next 🡨 pt

End if

1. Assume a queue containing one or more nodes maintained as a singly linked list with a list head. The name and age fields of the head node normally contain information on the contents of the list. The boundary condition for an empty queue is F = Ω and setting R 🡨 Location(F). Write an algorithm to return all nodes in the list to the available storage pool **one node at a time** starting at the left and working to the right. A sample list follows:

R

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pt | // | // | F = 7 |  | Sam | 35 | 968 |  | May | 47 | 45 |  | Hope | 8 | Ω |

10 7 986 45

The internal node format is:

|  |  |  |
| --- | --- | --- |
| Name | Age | Next |

The Name and age fields of the “list head” at location 10 are not currently utilized.

1. Assume a data structure pointed to by Pt containing one or more nodes maintained as a singly linked list. The boundary condition for an empty list is Pt = Ω. Write an algorithm to return all nodes in the list to the available storage pool in the most efficient (fastest) fashion possible. A sample list follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | PT |  | Sam | 35 | 968 |  | May | 47 | 45 |  | Hope | 8 | Ω |

7 986 45

The internal node format is:

|  |  |  |
| --- | --- | --- |
| Name | Age | Next |

The Name and age fields of the “list head” at location 10 are not currently utilized.

Hint:

If (pt = Ω )

Null

Else

Front 🡨 pt

While (pt.next <> Ω) loop

Pt 🡨 pt.next

End loop

Pt.next 🡨 avail

Avail 🡨 front

Pt 🡨 Ω

End if

1. Given the following declaration, write a formula to determine the exact memory location of any random entry in the array A[ M, N, P, S, T, W ] using lexicographic order for storage. Assume an inventory record contains 200 characters and the array starts at location “base.” **YOU MUST CLEARLY and NEATLY SHOW YOUR WORK TO RECEIVE CREDIT**!

A: Array( -20 .. 9, 5 .. 8, ‘A’ .. ‘E’, 0.. 4, -4 .. 0, ‘C’ .. ‘F’ ) of InventoryRecord.

Hint: d1 .. d6 🡨 30, 4, 5, 5, 5, 4

A0 is base, a6 🡨 1, a5 🡨 4, a4 🡨 5\*4 … a1 🡨 4\*5\*5\*5\*4 c 🡨 200

Loc{ A[M, N, P, S, T, W] }

= a0 + 200 \* { a1(M-20) + a2\*(N-5) + a3 \* (P-‘A’) + a4\* S + a5 \* (T+4) + a6\*(W-‘C’) }

1. Write an algorithm to efficiently concatenate the list point to by PT1 to the right of the list pointed to by PT2. PT1 should be set to omega when the concatenation operation is complete. Be careful, this is a different algorithm than the one developed in class.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Tom |  |  | Sam |  |  | Betty |  |  | PT1 |
|  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Barbie |  |  | Ken |  |  | Sally |  |  | PT2 |
|  |  |  |  |  |  |  |  |  |  |  |

For the example above the result would appear as:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Tom |  |  | Sam |  |  | Betty |  |  | PT1🡨 Ω |
|  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Barbie |  |  | Ken |  |  | Sally |  |  | PT2 |
|  |  |  |  |  |  |  |  |  |  |  |

Hint (does not cover either list being empty):

T 🡨 Pt2.link

Pt2.link 🡨 Pt1.link

Pt1.link 🡨 T

Pt2 🡨 Pt1

Pt1 🡨 Ω

1. Choice (do only one of the following):
2. Discuss the division remainder technique in great detail.
3. Discuss collision handling using the linear probe technique in great detail.
4. We are evaluating the garbage collection algorithms for three programming languages. Language L1 satisfies memory request by sequentially allocating memory from the heap till memory is exhausted. When memory is exhausted, it halts the program in execution to compact unused memory at the top forming a hole allowing execution to resume (essentially like Java/Python). Languages L2 and L3 allow programmers to request memory dynamically also. It is the programmers responsibility however to reclaim memory. Assume two different types of objects with different space requirements. All objects of the same type require the same amount of memory. Language L2 places all freed blocks on the same (single) available storage list. Language L3 places blocks of the same type on separate available storage list. Returned memory is used prior to allocating new memory from the heap in both languages L2 and L3. New memory is obtained from the heap using the algorithm developed in class for new memory request. Compare and contrast the advantages and disadvantages of the memory schemes for all three languages.
5. Assume a doubly linked circular list of airline reservations in ascending lexicographic order by customer name with a list head at location A, e.g., Pt 🡨 A:

Pt

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | Flt105 Texas | B |  | A | Bob | C |  | B | Tom | A |  |
|  | A |  |  |  | B |  |  |  | C |  |  | |

The customer node format is:

|  |  |  |
| --- | --- | --- |
| Left Link | Information | Right Link |

The list head contains the flight number and destination. Nodes in the list represent passengers. The empty list is represented by:

|  |  |  |
| --- | --- | --- |
| A | Flt105 Texas | A |
|  | A |  |

**Write an algorithm to insert a new node pointed to by IP to the left of the node pointed to by Pt2. Your algorithm should cover all cases!** A sample insertion follows.

Prior to insertion

Pt

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | Flt105 Texas | B |  | A | Bob | C |  | B | Tom | A |  |
|  | A |  |  |  | B |  |  |  | C |  |  | |

After insertion : Pt Pt2

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | Flt105 Texas |  | B |  | A | Bob | G |  | G | Tom | A |  |
|  | A |  |  |  |  | B |  |  |  | C |  |  | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | B | Sara | C |  |  |  |

IP G

Hint: IP.RL 🡨 Pt2; IP.LL 🡨 Pt2.LL; (IP.RL).RL 🡨 IP; Pt2.LL 🡨 IP;

I have completed the first pass creating Test 2. Be sure you have mastered topological sorting, queues kept as linked list with the front pointer stored in the link field of the head node, doubly linked list, garbage collection, operations on circular lists, multidimensional arrays, sparse arrays, triangular arrays, storing nodes in specific orders such as ascending or descending order. PLEASE READ ALL QUESTIONS CAREFULLY NOT JUMPING TO CONCLUSIONS AS TO WHAT IS DESIRED!

I have completed the first pass creating Test 2.  Be sure you have mastered topological sorting, queues kept as linked list with the front pointer stored in the link field of the head node, doubly linked list, garbage collection, operations on circular lists, multidimensional arrays, sparse arrays, triangular arrays, storing nodes in specific orders such as ascending or descending order.  PLEASE READ ALL QUESTIONS CAREFULLY NOT JUMPING TO CONCLUSIONS AS TO WHAT IS DESIRED!  Assume a 7:30 start as long as it is convenient for everyone.  If not convenient, please let me know ASAP so that I may notify everyone in a timely manner of the 8:00 start time.  You should have no trouble completing the test in less than an hour and 20 minutes.

4) In class we developed a dynamic storage allocator using the data structure below. A priority goal was memory reuse. **Write the algorithms as developed in class to allocate and reclaim storage assuming all requests for memory are of size “C.”**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| L0 |  |  | PoolMax |  | SeqMin |  |
| Used dynamic linked space | | | Free dynamic link space | | Dynamic sequential structures | |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Node format  on Avail list: | |  | Not used | Link |  |  |

Avail 🡨 Ω, if there are no currently available dynamically allocated nodes for reuse of the proper size. Otherwise Avail points to the first available node on the reuse list.

5) Given the following declarations, write a formula to calculate the location of any random element in the array. You need not do the actual multiplies in the final formula but must clearly evaluate the size of all dimensions (di) and evaluate all ai in the equations. You need not do the actual multiplies as long as all values are clearly and neatly shown. Assume the translator numbers all programmer defined enumeration types having n values internally as integers from 0 to n-1. Please assume an integer requires 4 bytes of storage. To receive credit, your answer must be neat and in the proper logical order as developed in class.

type fruit = (banana, apple, pear, grape, orange, pineapple, cantaloupe, watermelon);

hope: array[ -10..12, pear .. pineapple, 11..47, 0..146, fruit] of integer.

1. Develop an addressing scheme to find the location of any random A(row, col) in a lower triangular matrix:

Note we only need to store the entries A(row, col) for 0 <= row <= col <= n. Assume each entry in the array requires 5 units of storage.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A(0,0) |  |  |  |
|  | A(1,0) | A(1,1) |  |  |
|  | °°° |  |  |  |
|  | A(n,0) | A(n,1) | °°° | A(n,n) |

1. Assume a singly linked circular list pointed to by Pt. **Write an algorithm to return all nodes to the available storage pool pointed to by Avail in the most efficient fashion.**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Tommy |  |  | Betty |  |  | Adam |  |  | Pt |
|  |  |  |  |  |  |  |  |  |  |  |

1. Assume a doubly linked circular list of airline reservations in ascending lexicographic order by customer name with a list head at location A, e.g., Pt 🡨 A:

Pt

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Flt105 Texas | B |  | A | Bob | C |  | B | Tom | A |  |
|  | A |  |  |  | B |  |  |  | C |  |  | |

The customer node format is:

|  |  |  |
| --- | --- | --- |
| Left Link | Information | Right Link |

The list head contains the flight number and destination. Nodes in the list represent passengers. The empty list is represented by:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | Flt105 Texas | | A | |
|  | A |  | |  |

**Write an algorithm to insert a new node pointed to by IP as the first node to the right of the list head. Your algorithm must work if the list is initially empty.**

1. Assume the existence of two linked list pointed to by Pt1 and Pt2 respectively. Write an algorithm to concatenate the list pointed to by Pt2 to the right of the list pointed to by Pt1. Pt2 should be set to empty after the concatenation operation. Your algorithm must cover all special cases such as either Pt1, Pt2 or both being empty list. A sample follows:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Before:** |  |  |  |  |  |  |  |  |
| Pt1 |  | 8 |  |  | Jill | Ω |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Pt2 |  | Sam |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **After:** |  |  |  |  |  |  |  |  |  |
| Pt1 |  | 8 |  |  | Jill |  |  | Sam | Ω |
|  |  |  |  |  |  |  |  |  |  |
| Pt2 🡨 Ω |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Partial answer (ignore special cases);

p 🡨 pt1

while p /= Ω loop

p 🡨 p.link

end while

p.link 🡨 pt2

pt2 🡨 Ω

1. Assume a queue maintained as a linked list. The boundary condition for an empty queue is F := Ω and setting R := Location(F). Write an algorithm to delete the node at the head of the list observing all boundary conditions including underflow. The deleted node should be returned to the available storage pool after the contents are processed. The node format is an “info” field followed by a “link” field.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sample: |  | info | link |  |  |  | R |
|  | F |  | 8 |  |  | Jill | Ω |  |

1. Write an algorithm (not code) using the proper notation to solve the following problem using the strategy developed in class to sort related records into groups. Points will be deducted for use of code.

**Mission Impossible - Sorting:**

Job

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Sally |  |  | 8 | Ω |  |  |
| pgmr |  |  |  |  |  |  |  |  |  |
|  | Ω |  |  |  |  |  |  |  |  |
| acct |  |  |  |  |  |  |  |  |  |
|  |  |  | Tim |  |  | 1 | Ω |  |  |
| mgr |  |  |  |  |  |  |  |  |  |
|  |  |  | Sam | Ω |  |  | Tim | mgr |  |
| sales |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | ○○○ |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | Sam | Sales |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | Sally | pgmr |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Records |  |  | 1 | mgt |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 8 | pgmr |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

1. We have created the following list containing “list head” pointers to two consecutive nodes allowing efficient traversal of the list in both directions using the XOR technique to build the link addresses. Develop an algorithm to traverse the list from the head nodes listing all passengers by ticket (print the info fields) from left to right in the diagram. You must allow for an empty list. The maximum number of passengers in the list is not known prior to attempting the list traversal.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **P1** |  |  | **P2** |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | info | **LA** |  | info | **LB** |  | info | **LC** |  | info | **LD** |  | info | **LE** |
|  | **A** |  |  | **B** |  |  | **C** |  |  | **D** |  |  | **E** |  |

FLT 747 LAX 2:00 A.M. ticket 1 ticket 2 ticket 3

P1 and P2 are treated as list heads for the flight – other nodes represent passengers.

1. Arrays and linked list typically are homogeneous. Explain in great detail how to create sequentially allocate arrays and linked list consisting of heterogeneous entries! List all advantages and disadvantages of the technique. Use appropriate diagrams to illustrate your response.

4) Process the following precedence relations using the Topological Sort Algorithm taught in class. **You must do both parts “A” and “B” to receive credit**.

1. **First process all relations in the order specified below to build the initial data structure**. Note the subscripts and QLinks are the names of the persons responsible for the task. Label your answer as “Part A” using the following format. **You must use all the relations in the indicated order.**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QLink | 1 | Sam | Sara | Tim | Tom | Jill | Bob | 8 | 9 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω |

Jill < Bob, 9 < 1, Sam < 1, 8 < Jill, 9 < Tom, Sam < Tom, Sara < Tom, 8 < Jill, Sam < Jill, 1 < Tom, Sara < Bob

Hint: partial final queue:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QLink | 1 | Sam | Sara | Tim | Tom | Jill | Bob | 8 | 9 |
| Sam | Bob | Sara | Tim | 8 | 0 | 1 | Tom | 9 | Jill |
|  | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω |

1. ***Make a copy of the data structure produced in Part A and label it “Part B*.” Utilize the count fields to build the linked allocated queue utilized by our algorithm. Complete the topological sort. The count fields of all tasks should be decremented appropriately and if any go to zero, they should be utilized as links to implement the queue** (all link pointers adjusted). If a loop is encountered (there is no solution), state which tasks form the loop.
2. Assume the existence of two doubly linked list pointed with logical pointers Pt1 and Pt2 respectively. Write an algorithm to concatenate the list pointed to by Pt2 to the right of the list pointed to by Pt1. Pt2 should be set to empty after the concatenation operation. Your algorithm must cover all special cases such as either Pt1, Pt2, or both being empty list. A sample follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | **Before:** |  |  |  |  |  |  |  |  |  |  |  |
|  | 25 | Pt1 | 5 |  | 50 | 8 | 25 |  | 5 | Jill | 50 |  |  |  |  |
|  | 50 |  |  |  | 5 |  |  |  | 25 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 12 | Pt2 | 12 |  | 10 | Sam | 10 |  |  |  |  |  |  |  |  |
|  | 10 |  |  |  | 12 |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **After:** |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 12 | Pt1 | 5 |  | 50 | 8 | 25 |  | 5 | Jill | 12 |  | 25 | Sam | 50 |
|  | 50 |  |  |  | 5 |  |  |  | 25 |  |  |  | 12 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 10 | Pt2 | 10 |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Partial answer ignoring special cases:

(pt1.llink),rlink 🡨 pt2.rlink

(pt2.rlink).llink 🡨 pt1.llink

{pt2.llink).rlink 🡨 pt1

pt1.llink 🡨 pt2.llink

pt2.llink 🡨 pt2.rlink 🡨 Ω

2) Assume a singly linked circular list pointed to by Pt. **Write an algorithm to insert a new node pointed to by IP in decending (lexicographic) order.**  You must cover all possible cases including the empty list. Be sure your algorithm checks for overflow.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Tommy |  |  | Betty |  |  | Adam |  |  | Pt |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | IP |  | 1 |  |  |  |

I checked for 4 cases: notes page 64 to insert new first element, empty list, and new last element. You had to insert the interior node (close example on page 46)

3) Assume a queue maintained as a linked list. The boundary condition for an empty queue is F := Ω and setting R := Location(F). Write an algorithm to return all nodes in the list to the available storage pool.

Hint:

If (F = Ω)

Nothing to return

Else

R.next <- Avail

F <- Ω

R <- loc(F)

End if

4) Given the following declarations, write a formula to calculate the location of any random element in the array. You need not do the actual multiplies in the final formula but must clearly evaluate the size of all dimensions (di) and evaluate all ai in the equations. You need not do the actual multiplies to calculate each ai as long as all values are clearly and neatly shown. Assume the translator numbers all programmer defined enumeration types having n values internally as integers from 0 to n-1. Please assume an integer requires 4 bytes of storage.

type fruit = (banana, apple, pear, grape, orange, pineapple, cantaloupe);

Hope: array[ -12..12, apple .. grape, 0..47, 17..146, banana .. cantaloupe] of integer.

Hint:

Loc( Hope[I, J, K, L, M] ) = a0 + ( a1\*(I+12) + a2\*(J-1) + a3\*K + a4\*(L-17) + a5\*M ) \* 4

D1 = 25, d2 = 3, d3 = 48, d4 = 130, d5 = 7

a0 = base address of array, a5 = 1, a4 = 7\*1, a3 = 130\*7, a2 = 48 \* 130 \* 7, a1 = 3 \* 48 \* 130 \* 7

5) In class we developed a sorting algorithm to sort secret agents by job category requiring a single access to each record. Assume the array “Job” with an entry for each of N job categories. Further assume raw input records contain a person’s name and job category. Elements of the linked list (nodes) contain only the person’s name and a pointer to the next person having the same job category or an Ω to indicate the end of list. The array “Job” is to be treated as a set of list heads for a stack. **Write the sorting algorithm!**

**Mission Impossible - Sorting:**

Job

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | Sally |  |  | 8 | Ω |  |  |
| pgmr |  |  |  |  |  |  |  |  |  |
|  | Ω |  |  |  |  |  |  |  |  |
| acct |  |  |  |  |  |  |  |  |  |
|  |  |  | Tim |  |  | 1 | Ω |  |  |
| mgr |  |  |  |  |  |  |  |  |  |
|  |  |  | Sam | Ω |  |  | Tim | mgr |  |
| sales |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | ○○○ |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | Sam | Sales |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | Sally | pgmr |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Records |  |  | 1 | mgt |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 8 | pgmr |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

See notes page 36-47 for multiple solutions.

6) Given the following declarations, write a formula to calculate the location of any random element in the array. You need not do the actual multiplies in the final formula but must clearly evaluate the size of all dimensions (di) and evaluate all ai in the equations. You need not do the actual multiplies as long as all values are clearly and neatly shown. Assume the translator numbers all programmer defined enumeration types having n values internally as integers from 0 to n-1. Please assume an integer requires 4 bytes of storage.

Type fruit = (banana, apple, pear, grape, orange, pineapple, cantaloupe);

Hope: array[ -12..12, apple .. grape, 0..47, 17..146, banana .. cantaloupe] of integer.

4) Assume a linked list with pointers PT1 and PT2 to two consecutive list heads. The list heads record the airline flight number and destination respectively. The sample in the diagram exhibits what the list looks like for Flight 105 to SHSU with one passenger Tom. “xor” means to “exclusive or” the indicated addresses. The links below are LA = C xor B, LB = A xor C, and LC = B xor A.

PT1 PT2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Flt105 | LA |  | SHSU | LB |  | Tom | LC |  |  |
|  | A |  |  | B |  |  | C |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

**Given pointers to two consecutive list heads (flight number and destination) in the list (left to right) PT1 and PT2, count and print the number of nodes in the list. Do not damage the list. PT1 and PT2 should never be moved!**. Cover all cases. The list heads at location A and B may never be deleted. The empty list is represented by:

PT1 PT2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Flt105 | LA = B xor B |  | SHSU | LB= A xor A |  |  |  |  |
|  | A |  |  | B |  |  |  |  |  |

Hint:

Knt <-- 0;

X <-- Pt1;

Y <-- Pt2;

If(Pt1.Link = 0] {list is empty]

Print(Knt);

Else

While( Y /= Pt1) loop

Knt <-- Knt + 1;

Temp <-- Y

Y <-- Y.Link xor X

X <-- Temp

End loop

Print( Knt – 1);

End if;

5) We are evaluating the garbage collection algorithms for three programming languages. Language L1 satisfies memory request by sequentially allocating memory from the heap till memory is exhausted. When memory is exhausted, it halts the program in execution to compact unused memory at the top forming a hole allowing execution to resume (essentially like Java/Python). Languages L2 and L3 allow programmers to request memory dynamically also. It is the programmers responsibility however to reclaim memory. Assume two different types of objects with different space requirements. All objects of the same type require the same amount of memory. Language L2 places all freed blocks on the same (single) available storage list. Language L3 places blocks of the same type on separate available storage list. Returned memory is used prior to allocating new memory from the heap in both languages L2 and L3 for new memory request. Compare and contrast the advantages and disadvantages of the memory schemes for all three languages.

6) Discuss the square and extract n bits technique with respect to hashing in detail. Explain its advantages. Your explanation should include any special considerations for numeric data, character data, and transpositions of ASCII characters (e.g., “AB” versus “BA”).

Test two concentrates on dynamically allocated memory. We covered singly linked, doubly linked, how to traverse linked list in both directions (inefficiently and efficiently), creation and traversal of list by manipulating the binary representation of the physical addresses (XOR), topological sorting, garbage collection, random insertion, random deletion, insertion in lexicographic order, writing dynamic storage allocators, how to reclaim/organize returned dynamic memory, circular list, sorting records sharing a common key into groups, stacks, queues, and deques.

Do not embarrass either of us by coming to the test unable to perform a topological sort following the algorithm taught in class or unable to insert new nodes in lexicographic order. Anticipate one or more question allowing you to extend your knowledge to a new application area/problem.

The test instructions follow:

**CS362 Data Structures Test 2 Spring 2010 Burris**

Answer four questions **including questions 1 and 2**. Number your questions on the answer sheets in ascending numeric order from one through six 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. Start each question you answer on a new page. You may delete multiple questions on the same page. 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.

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. From a professional standpoint, you should be able to answer every question. After the test, please review any material you feel needs more work.

5) In class we developed a dynamic storage allocator using the data structure below. A pri1 goal was memory reuse. **Write the algorithms as developed in class to allocate and reclaim storage assuming all requests for memory are of size “C.”**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| L0 |  |  | PoolMax |  | SeqMin |  |
| Used dynamic linked space | | | Free dynamic link space | | Dynamic sequential structures | |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| Node format  on Avail list: | |  | Not used | Link |  |  |

Avail := Ω, if there are no currently available nodes, otherwise Avail points to the first available node on the list.

1. Write an algorithm to add a new node in “**descending**” lexicographic order by name to a singly linked list. A sample list appears below. Your algorithm must cover every possible case. Assume the list is empty if Pt 🡨 Ω.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pt | Tom. | 27 |  |  | Sam | 35 |  |  | May | 47 |  |  | Hope | 8 | Ω |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

The node format is:

|  |  |  |
| --- | --- | --- |
| Name | Age | Next |

1. Assume a queue containing one or more nodes maintained as a singly linked list. The boundary condition for an empty queue is F := Ω and setting R := Location(F). Write an algorithm to return all nodes in the list to the available storage pool in the most efficient fashion possible. A sample list follows: R

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pt | // | // | F = 7 |  | Sam | 35 | 968 |  | May | 47 | 45 |  | Hope | 8 | Ω |

10 7 986 45

The internal node format is:

|  |  |  |
| --- | --- | --- |
| Name | Age | Next |

The Name and age fields of the “list head” at location 10 are not currently utilized.

Hint: Since it is a queue you have a rear pointer, R. The value of F is Pt.next. Partial solution:

If( pt.next /= Ω )

R.next 🡨 avail

Avail 🡨 pt.next

Pt.next <- Ω // boundary condition for empty list

R 🡨 pt // boundary condition for empty list

End if

1. Assume a singly linked circular list pointed to by Pt. **Write an algorithm to insert a new node pointed to by IP in descending lexicographic order**.. You must cover all possible cases including the empty list. Be sure your algorithm checks for overflow.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Terry |  |  | 1 |  |  | Adam |  |  | Pt |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  | IP | Sara |  |  |  |  |  |  |  |

Choice (do only one of the following):

A) Assume a doubly linked circular list of airline reservations in ascending lexicographic order by customer name with a list head at location A, e.g., Pt 🡨 A:

Pt

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Flt105 Texas | B |  | A | Bob | C |  | B | Tom | A |  |
|  | A |  |  |  | B |  |  |  | C |  |  | |

The customer node format is:

|  |  |  |
| --- | --- | --- |
| Left Link | Information | Right Link |

The list head contains the flight number and destination. Nodes in the list represent passengers. The empty list is represented by:

|  |  |  |
| --- | --- | --- |
| A | Flt105 Texas | A |
|  | A |  |

**Write an algorithm to insert a new node pointed to by IP in the list in ascending lexicographic order. You may never delete the list head at location A.**

3)

4) Write an algorithm to concatenate the string pointed to by Pt2 to the end of the string pointed to by Pt1. In the example below, the result would be Betty, 1, Zoey, Bob, followed by Sam. When finished, Pt1 will point to the combined list and Pt2 = Ω. Be user to cover all cases such as either or both list being empty initially.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pt1 |  | Betty |  |  | 1 |  |  | Zoey | Ω |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Pt2 |  | Bob |  |  | Sam | Ω |  |  |  |

5) Assume a queue maintained as a linked list. The boundary condition for an empty queue is F := Ω and setting R := Location(F). Write an algorithm to delete items from the rear creating an input restricted deque. Observe all current boundary conditions.

4) Explain in detail why circular list (as shown) are superior to traditional singly linked list in most computer applications. When are they not superior?

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 8 |  |  | Sam |  |  | 1 |  |  | PT |
|  |  |  |  |  |  |  |  |  |  |  |

2) Choice (do only one of the following):

A) Compare and contrast square and extract N bits with the division remainder technique for hashing in great detail.

B) Discuss the Random Probe technique for collision handling in detail. Include how to know the table is full, conduct a successful search, unsuccessful search, whole record insertion, whole record deletion, space reclamation, and reducing the time to search for existing records when records they previously collided with are deleted.

3) Assume a doubly linked list with a list head as shown. The list head contains information on the flight including the flight number, destination, etceteras. The other nodes in the list represent customers. Write an algorithm to locate a random passenger in the list starting at the list head. If found, delete them from the list. If not found, print an appropriate message.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Head |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 12 | Flt | 40 |  | 10 | Bob | 12 |  | 40 | Tom | 10 |  |  |
|  |  | 10 |  |  |  | 40 |  |  |  | 12 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

5) Assume a queue maintained as a linked list. The boundary condition for an empty queue is F := Ω and setting R := Location(F). Write an algorithm to insert items at the front of the queue creating an output restricted deque. Observe all current boundary conditions.

3) Assume a singly linked circular list pointed to by Pt. **Write an algorithm to insert a new node (passenger) pointed to by IP in descending lexicographic order**.. You must cover all possible cases including the empty list. Be sure your algorithm checks for overflow. The list represents passengers traveling by plane (e.g., FLT21). When the plane lands, all passengers leave but the head node is retained for future flights.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Terry |  |  | 1 |  |  | FLT 21 |  |  | Pt |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  | IP | Sara |  |  |  |  |  |  |  |

Empty List:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | FLT 21 |  |  | Pt |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

1) Process the following precedence relations using the Topological Sort Algorithm taught in class. **You must do both parts “A” and “B” to receive credit**.

1. **First process all relations in the order specified below to build the initial data structure**. Note the subscripts and QLinks are the names of the persons responsible for the task. Label your answer as “Part A.”

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QLink | 1 | Sam | Sara | Tim | Tom | Jill | Bob | 8 | 9 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω |

1 < Tom, 8 < Tim, Tim < Tom, 9 < Jill, Sam < Jill, 8 < Tim, Bob < Jill, Sara < Tim, 1 < Sam, Tim < Bob

Partial answer:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QLink | 1 | Sam | Sara | Tim | Tom | Jill | Bob | 8 | 9 |
|  | 0 | 1 | 0 | 3 | 2 | 3 | 1 | 0 | 0 |
|  | | | | | | | | | Ω | Ω | | | | | | |

| | | | | | |

Sam Jill Tim Bob Jill Tim Jill

| | |

Tom Tom Tim

1. ***Make a copy of the data structure produced in Part A and label it “Part B*.” Utilize the count fields to build the linked allocated queue utilized by our algorithm. Complete the topological sort. The count fields of all tasks should be decremented appropriately and if any go to zero, they should be utilized as links to implement the queue** (all link pointers adjusted). If a loop is encountered (there is no solution), state which task form the loop.

Partial answer:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QLink | 1 | Sam | Sara | Tim | Tom | Jill | Bob | 8 | 9 |
| 1 | Sara | Tim | 8 | Bob | Jill | 0 | Tom | 9 | Sam |
|  | | | | | | | | | Ω | Ω | | | | | | |

| | | | | | |

Sam Jill Tim Bob Jill Tim Jill

| | |

Tom Tom Tim

Sorted: 1, Sara, 8, 9, Sam, Tim, Bob, Tom, Jill

Assume a singly linked list pointed to by Pt. Write an algorithm to insert a new node pointed to by IP in descending lexicographic order.. You must cover all 4 possible cases including the empty list. Be sure your algorithm checks for overflow. You have limited time. I suggest you write the algorithm for clarity rather than efficiency!

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pt |  | Sara |  |  | 1 |  |  | Adam | Ω |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | IP |  |  |  |  |  |  |  |  |

Partial Answer; This does not include the check for the empty list:

P1 <--P2<--Pt

While(p1<> null) and (IP.Info > P1.Info) loop

P2 <-- P1

P1 <\_\_ P1.link

End loop

If Pt = P1

IP.link <-- Pt

Pt <-- IP

Else

P2.Link <-- IP

IP.Link <-- P1

End if

2) Assume a singly linked circular list pointed to by Pt. **Write an algorithm to insert a new node pointed to by IP in descending lexicographic order**.. You must cover all possible cases including the empty list. Be sure your algorithm checks for overflow.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Terry |  |  | 1 |  |  | Adam |  |  | Pt |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  | IP | Sara |  |  |  |  |  |  |  |

Hint:

[empty list?]

If( Pt = Ω)

Pt <-- Ip.next <-- Ip

Else [new last node?]

If( Ip.info < pt.info)

Ip.next <-- Pt.next;

Pt.next <-- Ip;

Pt <-- Ip;

Else [new first node?]

P <-- Pt.next;

If(Ip.info > P.info)

Ip.next <-- P;

Pt.next <-- Ip;

Else [ new interior node, find location]

P1 <-- Pt.next;

P2 <-- Pt;

While( P1.info > IP,info) loop

P2 <-- P1;

P1t <-- P1.next;

End loop;

Ip.next <-- P1;

P2.next <-- Ip;

End if;

End if;

End if;

3) Write an algorithm to concatenate the list pointed to by Pt2 to the right of the list pointed to by Pt1. In the example below, the result would be Betty, 1, Zoey, Bob, followed by Sam. When finished, Pt1 will point to the combined list and Pt2 = Ω. Be sure to cover all cases such as either or both list being empty initially.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pt1 |  | Betty |  |  | 1 |  |  | Zoey | Ω |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Pt2 |  | Bob |  |  | Sam | Ω |  |  |  |

Hint:

If pt2 = Ω [no second list?]

Null;

Else

If pt1 = Ω [no first list?]

Pt1 <-- pt2;

Pt2 <-- Ω;

Else

[find end of list pointed to by pt1]

p <-- pt1;

While (p.next <> Ω ) loop

p <-- p.next;

End loop;

p.next <-- pt2;

pt2 <-- Ω

end if;

end if;

2) Assume a singly linked circular list pointed to by Pt. **Write an algorithm to insert a new node pointed to by IP as the last (right most) node in the list.** After the insertion in the sample list below, Adam will point to Betty and Pt will also point to Betty**.** You must cover all possible cases including the empty list. Be sure your algorithm checks for overflow.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Terry |  |  | 1 |  |  | Adam |  |  | Pt |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | IP |  | Betty |  |  |  |

5) We are evaluating the garbage collection algorithms for three programming languages. Language L1 satisfies memory request by sequentially allocating memory from the heap till memory is exhausted. When memory is exhausted, it halts the program in execution to compact unused memory at the top forming a hole allowing execution to resume (essentially like Java). Languages L2 and L3 allow programmers to request memory dynamically also. It is the programmers responsibility however to reclaim memory. Assume two different types of objects with different space requirements. All objects of the same type require the same amount of memory. Language L2 places all freed blocks on the same (single) available storage list. Language L3 places blocks of the same type on separate available storage list. Returned memory is used prior to allocating new memory from the heap in both languages L2 and L3 for new memory request. Compare and contrast the advantages and disadvantages of the memory schemes for all three languages.

4) Assume a linked list with pointers PT1 and PT2 to two consecutive list heads. The list heads record the airline flight number and destination respectively. The sample in the diagram exhibits what the list looks like for Flight 105 to SHSU with one passenger Tom. “xor” means to “exclusive or” the indicated addresses. The links below are LA = C xor B, LB = A xor C, and LC = B xor A.

PT1 PT2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Flt105 | LA |  | SHSU | LB |  | Tom | LC |  |  |
|  | A |  |  | B |  |  | C |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

**Given pointers to two consecutive list heads (flight number and destination) in the list (left to right) PT1 and PT2, count and print the number of nodes in the list. Do not damage the list. PT1 and PT2 should never be moved!**. Cover all cases. The list heads at location A and B may never be deleted. The empty list is represented by:

PT1 PT2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Flt105 | LA = B xor B |  | SHSU | LB= A xor A |  |  |  |  |
|  | A |  |  | B |  |  |  |  |  |

Hint:

Knt <-- 0;

X <-- Pt1;

Y <-- Pt2;

If(Pt1.Link = 0] {list is empty]

Print(Knt);

Else

While( Y /= Pt1) loop

Knt <-- Knt + 1;

Temp <-- Y

Y <-- Y.Link xor X

X <-- Temp

End loop

Print( Knt – 1);

End if;

1) Process the following precedence relations using the Topological Sort Algorithm taught in class. **You must do both parts “A” and “B” to receive credit**.

1. **First process all relations in the order specified below to build the initial data structure (algorithm steps 1& 2)**. Label the first part of your answer as “Part A.”

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| QLink | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω | Ω |

5<4, 6<9, 3<2, 5<7, 5<3, 4<6, 6<3, 6<1, 8<9, 7<3, 2<9, 5<4

1. ***Make a copy of the data structure produced in Part A and label it “Part B*.” Utilize the count fields to build the linked allocated queue utilized by our algorithm. Complete the topological sort. The count fields of all tasks should be decremented appropriately and if any go to zero, they should be utilized as links to implement the queue** (all link pointers adjusted). If a loop is encountered (there is no solution), state which task form the loop using the data structure.

3) Assume a doubly linked list with a list head as shown. The list head contains information on the flight including the flight number (FLT). The other nodes in the list represent customers. Write an algorithm to insert a new customer pointed to by IP in the list to the right of any specified node. You must locate the specified node to perform the insertion. If the specified customer does not exist, print an appropriate message. Be sure your algorithm covers all possible cases including the empty list.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Head |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 12 | Flt 2 | 40 |  | 10 | Bob | 12 |  | 40 | Tom | 10 |  |  |
|  |  | 10 |  |  |  | 40 |  |  |  | 12 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | IP |  |  | Tim |  |  |  |  |  |  |  |

Data Structures test 2 is scheduled for the week after spring break. I have completed the first pass making the test and anticipate the test will cover all material not covered on the first test (starts with dynamic storage allocation) up through and including the section titled “Indexing Sequentially Allocated Structures – Arrays.” I do not anticipate covering trees on test 2 at present.

Dr. B.

The current instructions follow:

**COSC 3319 Data Structures Test 2 Spring 2012 Burris**

Answer three questions **including questions 1**. Number your questions on the answer sheets in ascending numeric order from one through seven inclusive. Clearly write “Delete” on the answer sheets by the four questions you do not wish graded. Leave at least a one-inch margin at the top of every page. Start each question you answer on a new page. You may delete multiple questions on the same page. 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.

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. From a professional standpoint, you should be able to answer every question. After the test, please review any material you feel needs more work.

5) Assume a linked list with pointers PT1 and PT2 to two consecutive list heads. The list heads record the airline flight number and destination respectively. The sample in the diagram exhibits what the list looks like for Flight 105 to SHSU with one passenger Tom. “xor” means to “exclusive or” the indicated addresses. The links below are LA = C xor B, LB = A xor C, and LC = B xor A.

PT1 PT2

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Flt105 | LA |  | SHSU | LB |  | Tom | LC |  |  |
|  | A |  |  | B |  |  | C |  |  |  |
|  | Flight |  |  | Destination |  |  |  |  |  |  |

**Assume DP points to a random customer in the list. Write an algorithm to properly delete the customer pointed to by DP from the list given pointers to two consecutive list heads (flight number and destination) in the list (left to right) PT1 and PT2. Do not damage the list. PT1 and PT2 should never be moved!**. Cover all cases. The list heads at location A and B may never be deleted. The empty list is represented by:

PT1 PT2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Flt105 | LA = B xor B |  | SHSU | LB= A xor A |  |  |  |  |
|  | A |  |  | B |  |  |  |  |  |

6) Write an algorithm to concatenate the list pointed to by Pt2 to the right of the list pointed to by Pt1. In the example below, the result would be Betty, 1, Zoey, Bob, followed by Sam. When finished, Pt1 will point to the combined list and Pt2 = Ω. Be sure to cover all cases such as either or both list being empty initially.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Betty |  |  | 1 |  |  | Zoey |  |  | Pt1 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  | Bob |  |  | Sam |  |  |  |  |  | Pt2 |  |

1. Assume a queue maintained as a linked list. The boundary condition for an empty queue is F := Ω and setting R := Location(F). Write an algorithm to delete the node at the rear of the queue observing all boundary conditions including underflow. The deleted node should be returned to the available storage pool after the contents are processed. The node format is an “info” field followed by a “link” field. If you delete the last node in the list, be sure to observe all boundary conditions. Your algorithm will effectively create an input restricted deque.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Sample: |  | info | link |  |  |  | R |
|  | F |  | 8 |  |  | Jill | Ω |  |