|  |
| --- |
| Serial No: |
| **Final Term Exam** |
| **Total Time: 3 Hours** |
| **Total Marks: 100** |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Signature of Invigilator |

|  |
| --- |
| **CS2001 Data Structures** |
| Friday, December 23, 2022 |
| **Course Instructor** |
| Dr. Hashim Yasin  Dr. Anwar Shah  Mr. Muhammad Usman Joyia  Mr. Muhammad Yousuf |

|  |
| --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ . .  Roll No Section Signature |

## DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

**Instructions:**

1. Verify at the start of the exam that you have a total of Eight (08) questions printed on Nine (09) pages including this title page.
2. Attempt all questions on the question-book and in the given order.
3. The exam is closed books, closed notes. Please see that the area in your threshold is free of any material classified as ‘useful in the paper’ or else there may a charge of cheating.
4. Read the questions carefully for clarity of context and understanding of meaning and make assumptions wherever required, for neither the invigilator will address your queries, nor the teacher/examiner will come to the examination hall for any assistance.
5. Fit in all your answers in the provided space. You may use extra space on the last page if required. If you do so, clearly mark question/part number on that page to avoid confusion.
6. Use only your own stationery and calculator. If you do not have your own calculator, use manual calculations.
7. Use only permanent ink-pens. Only the questions attempted with permanent ink-pens will be considered. Any part of paper done in lead pencil cannot be claimed for checking/rechecking.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Q-1 | Q-2 | Q-3 | Q-4 | Q-5 | Q-6 | Q-7 | Q-8 | Total |
| **Total**  **Marks** | **15** | **10** | **10** | **20** | **12** | **08** | **10** | **15** | **100** |
| **Marks Obtained** |  |  |  |  |  |  |  |  |  |

**Vetted By: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Vetter Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **University Answer Sheet Required:** | **No** |  |  |  |  | **Yes** |  |

**Question No. 1 Stack ADT and Applications [15 Marks]**

**Part A (10):** Consider the usual algorithm to convert an infix expression to a postfix expression. Suppose that you have read 10 input characters during a conversion and stack now contains these symbols:

|  |
| --- |
| / |
| + |
| ( |
| \* |

Now, consider that you read and process the 11th symbol of the input. Considering the above-mentioned stack state and draw the updated states for all the cases mentioned below separately where the 11th symbol is:

1. A number { 5 }:
2. A left parenthesis { ( }:
3. A right parenthesis { ) }:
4. A minus sign { - }:
5. A division sign { / }:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | / | | + | | ( | | \* | | |  | | --- | | ( | | / | | + | | ( | | \* | |
| |  | | --- | | \* | | |  | | --- | | - | | ( | | \* | |
| |  | | --- | | / | | + | | ( | | \* | | |

**Part B (5):** Consider the following infix expression and convert it to polish notation using stacks. Clearly mention the all the steps using stacks to claim **ANY** marks.

**((A\*B/C)^D)\*(E+F)^2**

|  |  |
| --- | --- |
| **Sr. Symbol Stack Prefix**  **0 (**  **1 2 ( 2**  **2 ^ ( ^ 2**  **3 F ( ^ ( 2F**  **4 + ( ^ ( + 2F**  **5 E ( ^ ( + 2FE**  **6 ) ( ^ 2FE+**  **7 \* ( \* 2FE+^**  **8 D ( \* ( 2FE+^D**  **9 ^ ( \* ( ^ 2FE+^D**  **10 C ( \* ( ^ ( 2FE+^DC**  **11 / ( \* ( ^ ( / 2FE+^DC** | **Sr. Symbol Stack Prefix**  **12 B ( \* ( ^ ( / 2FE+^DCB**  **13 \* ( \* ( ^ ( \* 2FE+^DCB/**  **14 A ( \* ( ^ ( \* 2FE+^DCB/A**  **15 ) ( \* ( ^ 2FE+^DCB/A\***  **16 ) ( \* 2FE+^DCB/A\*^**  **17 ) 2FE+^DCB/A\*^\*** |

**Question No. 2 Queue ADT [10 Marks]**

Given a Queue data structure that supports standard operations like ***enqueue()*,** ***dequeue()*, *isFull(),* and, *isEmpty()***. The Queue ADT is given as follows for your reference.

|  |
| --- |
| class QueueADT{  private:  int \*queueArray; //Pointer to array implemented as Queue   int queueSize; //Total size of the Queue  int front; int rear; int numItems;  public:  QueueADT(int);  ~IntQueue();  bool isEmpty();  bool isFull();  bool enqueue(int);  int dequeue(); }; |

However, in this task you need to implement *Stack ADT* using only instances of *Queue ADT* operations allowed on the instances. To keep the task simple following *Stack ADT* functions are required to implement:

*bool push(int data) // push an element in the queue maintaining the stack order*

*bool pop(int&data) //pop an item from queue and return it.*

*Note: You are not allowed to change the functionality of Queue ADT. However, you can use multiple instances of queue object as per your logic.*

|  |  |
| --- | --- |
| ***bool push(int data)* {**  **}** | ***bool pop(int&data)*** |

**Question No. 3 Linked List [10 Marks]**

Given a singly linked list, rotate it to the right by ***k*** nodes, where ***k*** denotes the number of right shifts you need to perform.

|  |  |
| --- | --- |
| if(!head) return head;  int len = 1;  Node\* tail = head;  while(tail->next)  {  len++;  tail = tail->next;  }  tail->next = head;  k %= len;  Node\* newHead = NULL;  for(int i = 0; i < len - k; i++)  {  tail = tail->next;  }  newHead = tail->next;  tail->next = NULL;  return newHead;  } | if(head == NULL||head->next == NULL) return head;  for(int i=0;i<k;i++)  {  node\* temp = head;  while(temp->next->next != NULL)  temp = temp->next;  node\* end = temp->next;  temp->next = NULL;  end->next = head;  head = end;  }  return head;  } |

**Question No. 4 Hash Maps [20 Marks]**

Note: Put only the final values in the blank cells of the table. Use another side of the paper for rough work which carries no marks. More than 2 mistakes in filling the table will result in zero marks.

1. **[Linear Probing] (04)** Suppose that the following keys are inserted in the order

**A B C D E F G**

into an initially empty linear-probing hash table of size 7, using the following hash function:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Keys | A | B | C | D | E | F | G |
| Hash(key,7) | **3** | **1** | **4** | **1** | **5** | **2** | **5** |

What is the result of the linear-probing array? Assume that the array size is fixed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
|  |  |  |  |  |  |  |

1. **[Double Hashing] (10)** Load the keys **18, 26, 35, 9, 64, 47, 96, 36,** and **70** in this order, in an empty hash table of size **n = 13.** Use double hashing **hi(key) = [h(key) + i\*hp(key)]% n** with the first hash function: **h(key) = key % n** and the second hash function: **hp(key) = 1 + key % (n-1).** For your convenience, a few entries are already filled in the table.

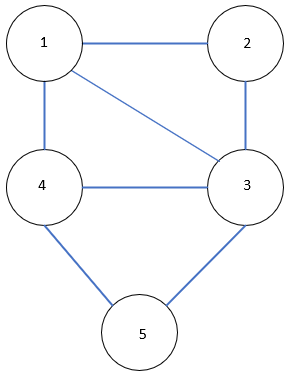
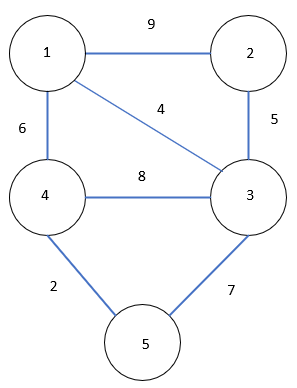
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| **26** |  |  |  |  | **18** |  |  |  | **35** |  |  |  |

1. **[Quadratic Probing] (06)** Load the keys **23, 13, 21, 14, 7, 8,** and **15**, in this order, in a hash table of size **7** using quadratic probing with **c(i) = ±i2** and the hash function: **h(key) = key % 7.** For your convenience, a few entries are already filled in the table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **21** |  | **23** |  |  |  | **13** |

**Question No. 5 Graphs ADT [12 Marks]**

A graph is a collection of edges and vertices, i.e., G = (V, E), where V is the set of vertices and E is the set of edges. We can represent a graph using an adjacency matrix, adjacency list, and compact list. Consider the following undirected graphs,



Graph A Graph B

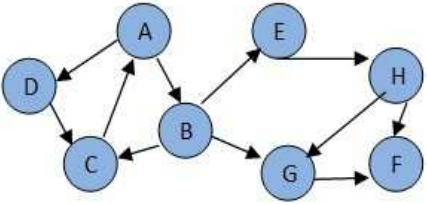
**Part A (04)**: Represent graph A and B using the compact list approach

|  |  |
| --- | --- |
|  |  |

**Part B (08):** Consider the following given graph and assume that there is a decision between multiple neighbor nodes in the BFS or DFS algorithms; we will always choose the neighbor node closest to the visiting node; for example, if we must visit P and R from Q, we will visit P first followed by R. Keeping this in mind, answer the following given the starting node A,

1. In what order will the nodes be visited using a breadth first search
2. In what order will the nodes be visited using a depth first search

Note: write the answers only. You can use the backside of this page for calculations.



|  |
| --- |
| **Breadth First Search** |
| **Depth First Search** |

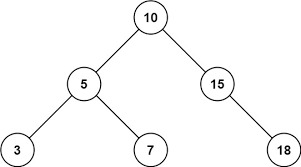
**Question No. 6 Binary Heaps [08 Marks]**

Heapify is the fast method to create a heap. Create a Max heap using heapify for the elements 5, 10, 30, 20, 35, 40, and 15. Clearly mark and write down all intermediatory steps. There will be no mark for direct answer.

|  |
| --- |
|  |

**Question No. 7 Binary Search Tree [02+05+03 = 10 Marks]**

Assume that you are writing a destructor for the BST ADT which in turn calls a function named as *destorySubTree(TreeNode \*node)* with tree root pointer as a value argument. You are required to complete the following recursive function strictly keeping the context in view. Tree traversals may be a bit helpful here. Identify which traversal will be directly applied and provide a strong justification for it. After that provide the code implementation with recursive call stack as well. a reference tree is given to use it for the call stacks.

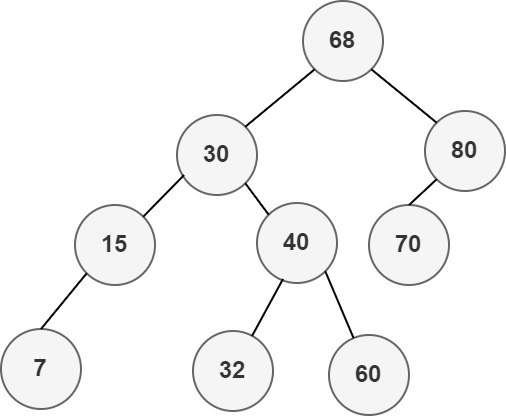


The function signature is given below:

|  |
| --- |
| **(02) Choose the appropriate Traversal and justify your rationale:**  Post Order Traversal is required because first destroy the left and right subtrees then parent itself. |
| **(05) Implementation**  void IntBST:: destroySubTree(TreeNode \*node){  // All the nodes of the tree must be deleted before returning to the caller function (i.e., Destructor)  If (node==NULL) return;  If(node->left != NULL)  destroySubTree(node->left);  If(node->right != NULL)  destroySubTree(node->right);  delete node;  } |
| **(03) Call Stack Explanation using the given tree:** |

**Question No. 8 AVL Tree [15 Marks]**

**Part A (03):** Consider the AVL Tree and calculate the balance factor of each node.



**Part B (06):** Consider the given AVL Tree again in **Part A**, Insert a new element **65**. Recalculate the BF and identify the imbalanced path if any and, make it balanced by performing the rotations.

**Note:** For rotations, construct the final AVL tree in the given box as under to get full marks. You can perform all steps involve in the process of rotations as **rough** work (if it is necessary) on the backside side of the paper but it contains **NO** marks. Rest of the operations should by clearly marked and mentioned.

|  |
| --- |
|  |

**Part C (06):** Consider the given AVL Tree in **Part A** once again, delete node **68** with appropriate node from right subtree. Recalculate the BF and identify the imbalanced path if any and, make it balanced by performing the rotations.

**Note:** For rotations, construct the final AVL tree in the given box as under to get full marks. You can perform all steps involve in the process of rotations as **rough** work (if it is necessary) on the backside side of the paper but it contains **NO** marks. Rest of the operations should by clearly marked and mentioned.

|  |
| --- |
|  |