# **Global Edge Software**

# **Training Department**

**Note :**

1. **All the assignments should be done using pointers , dynamic memory allocations .**
2. **All the programs should be written in the form of function.**
3. **All programs should follow ANSI C standards and Linux coding style**
4. ***Use -Wall option while compiling to arrest any warnings.***
5. Implement a Priority Library with the following functions available for the application to use. Write an application to test the library.
   1. Pri\_Create( ); - Returns a head pointer to the Pri\_queue
   2. Pri\_Insert(head, priority, data);
   3. Pri\_Delete(head, data);
   4. Pri\_Display(head);
   5. Pri\_Search(data); - Displays the queue number if data found, error otherwise.
   6. Pri\_Terminate(head); - delete the queue.
6. Implement a SLL (Singly Linked List) library with the following functions, which takes pointer to a void data type, so you can make a linked list from any kind of data. Its required that you have to write your own function to free your data. You will pass a pointer to your function while calling SLL free functions.
   1. *void initlist ( SLL \*\*list)*

Initialize \*list by setting the list pointer to NULL.

* 1. *SLL allocateNode (void \*data)*

Allocate space for a new node and initialize the data fields. The pointer to data can be anything. If a large amount data needs to be used use a structure.

* 1. *void appendNode ( SLL \*\*head, SLL \*\*new)*

Appends a node to the end of a list. head lists gets modified and new is the node to be appended.

* 1. *void insertNode (SLL \*\*head, SLL \*\*new)*

Insert a node at the beginning of a list. Insert node new at the beginning of list head. Note, the list head gets modified.

* 1. *void insertNodePos (SLL \*\*head, SLL \*\*new, int pos);*

Insert a node at the given position in the list. Here also list head gets modified.

* 1. *void delNode (SLL \*\*head, SLL \*node)*

Deletes a node from the list. head is the list to modify and node is the node to remove. Note, the user data is not freed by this routine.

* 1. *void freeNode (SLL \*\*list)*

Free a node. It does not do free the data. It checks before freeing if the list is NULL or not.

* 1. *void destoryNode (SLL \*\*head, SLL \*node, void (\*freeFunc)(void \*\*));*

Frees memory allocated for a node and the data associated with the node. The caller is responsible to write the function to free the memory for the data. The caller function frees the user data and the function *delNode* is used to delete the node. *delNode* calls *freeNode* to free the memory associated with the node.

* 1. *SLL \*getNthNode (SLL \*head, int n)*

Returns the nth node in a list. Note a node starts at 1. If there is no such node, NULL is returned.

* 1. int numNodes (SLL \*\*head)

Returns number of nodes in the list.

* 1. int emptyList (SLL \*list)

Checks if list is empty. Returns 1 if the list is empty, 0 otherwise.

* 1. void sortList (SLL \*lsit)

Sort the SLL either ascending or descending order.

* 1. void reverseList (SLL \*list)

Reverse the SLL.