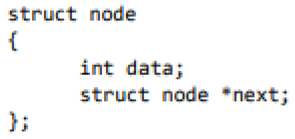
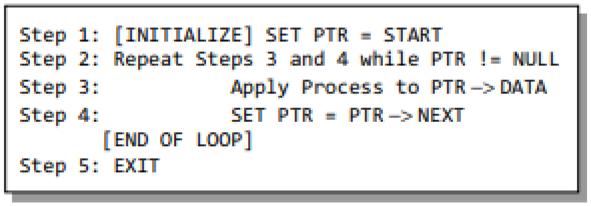
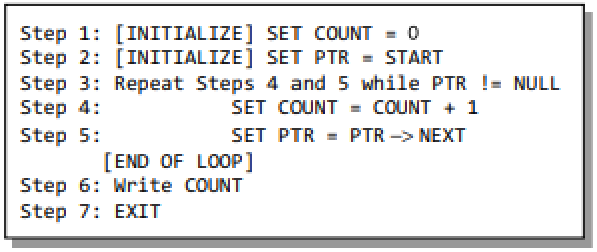
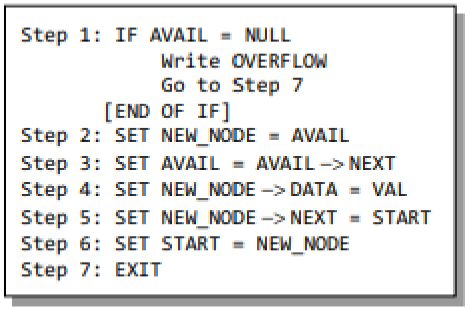
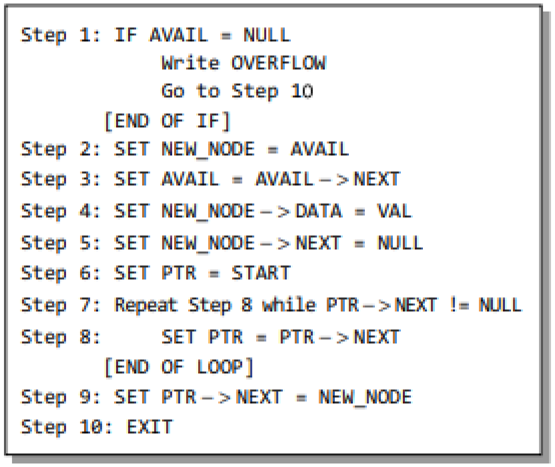
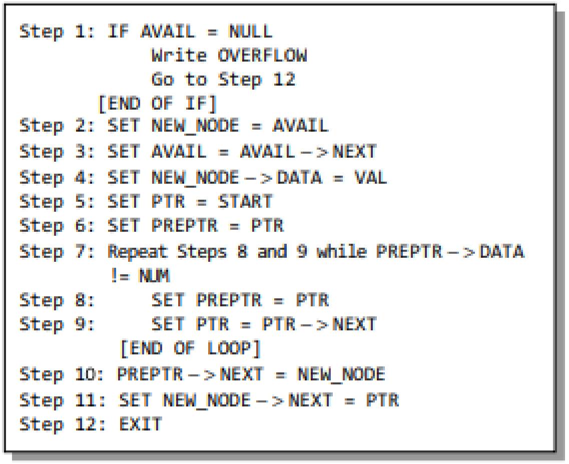
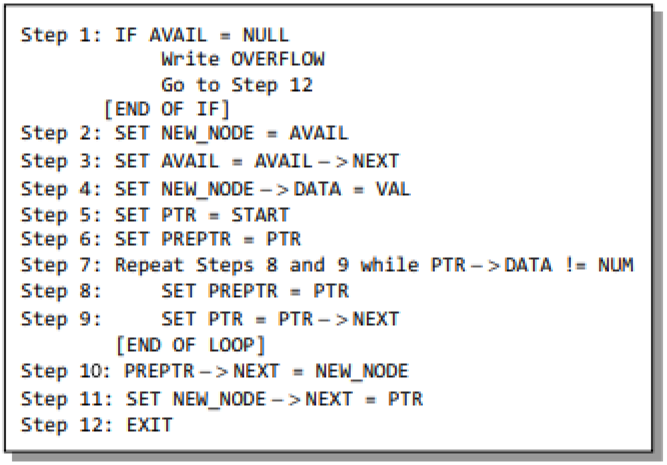
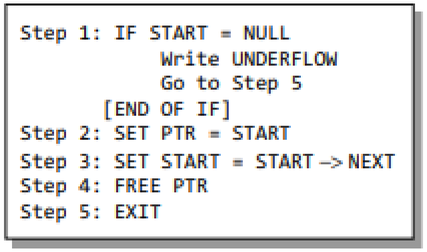
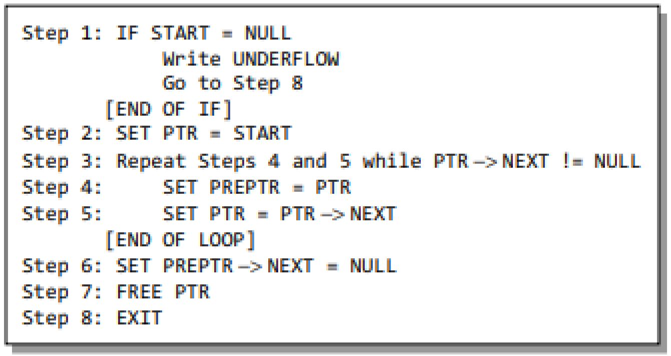
Module 1: Linked List Types, operations and Pseudocode

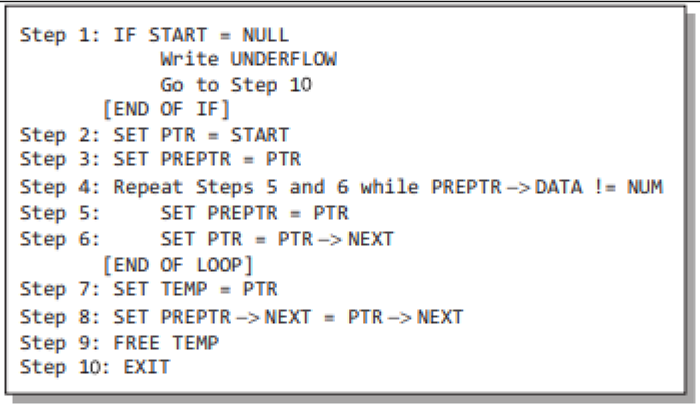
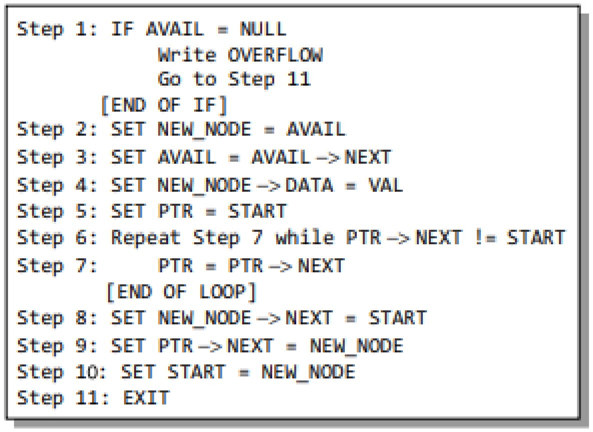
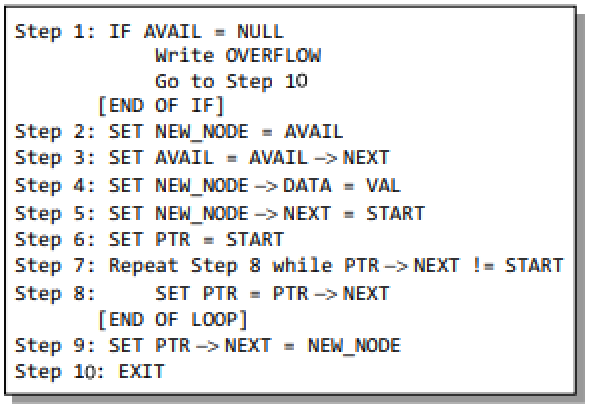
Data Structure types	Structure to be defined	Operations	Pseudocode

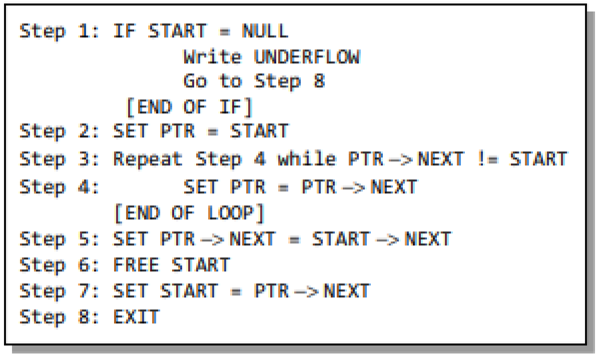
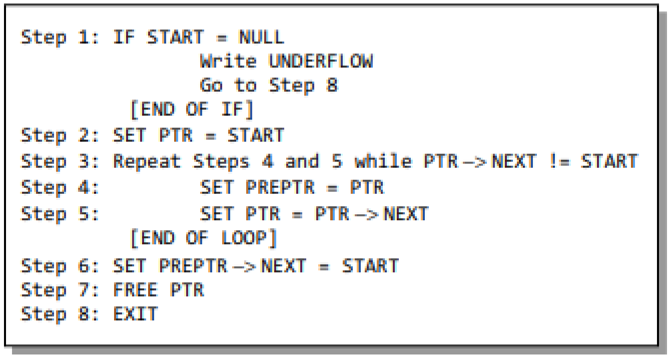
A. Singly Linked list	
	struct node
	{
	int data;
	struct node *next;
	};
		1. Traversing a list	
			Step 1: [INITIALIZE] SET PTR = START
			Step 2: Repeat Steps 3 and 4 while PTR != NULL
			Step 3: Apply Process to PTR -> DATA
			Step 4: SET PTR = PTR -> NEXT
			[END OF LOOP]
			Step 5: EXIT
			Figure 6.8 Algorithm for traversing a linked list

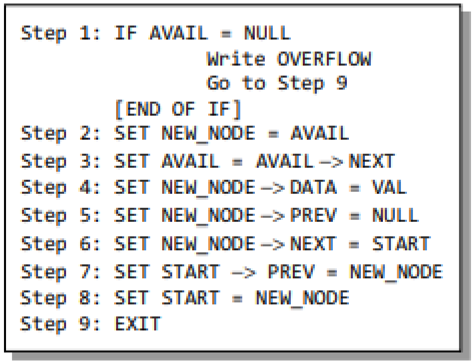
Singly Linked list		2. Print number of nodes(counting)	 Step 1: [INITIALIZE] SET COUNT = 0 Step 2: [INITIALIZE] SET PTR = START Step 3: Repeat Steps 4 and 5 while PTR != NULL Step 4: SET COUNT = COUNT + 1 Step 5: SET PTR = PTR -> NEXT [END OF LOOP] Step 6: Write COUNT Step 7: EXIT **Figure 6.9** Algorithm to print the number of nodes in a linked list
Singly Linked list		3. Insertion at the beginning	 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 7 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET NEW_NODE -> NEXT = START Step 6: SET START = NEW_NODE Step 7: EXIT **Figure 6.13** Algorithm to insert a new node at the beginning
Singly Linked list		4. Insertion at the end	

			 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 10 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET NEW_NODE -> NEXT = NULL Step 6: SET PTR = START Step 7: Repeat Step 8 while PTR -> NEXT != NULL Step 8: SET PTR = PTR -> NEXT [END OF LOOP] Step 9: SET PTR -> NEXT = NEW_NODE Step 10: EXIT **Figure 6.15** Algorithm to insert a new node at the end
Singly Linked list		5. Insertion in the middle after a given number	 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 12 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET PTR = START Step 6: SET PREPTR = PTR Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA != NUM Step 8: SET PREPTR = PTR Step 9: SET PTR = PTR -> NEXT [END OF LOOP] Step 10: PREPTR -> NEXT = NEW_NODE Step 11: SET NEW_NODE -> NEXT = PTR Step 12: EXIT **Figure 6.16** Algorithm to insert a new node after a node that has value NUM
Singly Linked list		6. Insertion in the middle before a given number	

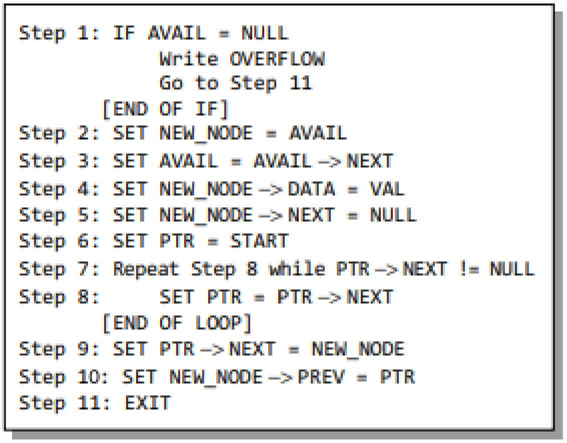
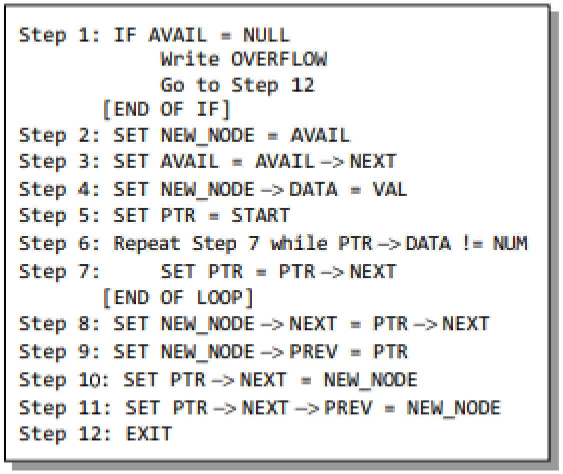
			 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 12 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL->NEXT Step 4: SET NEW_NODE->DATA = VAL Step 5: SET PTR = START Step 6: SET PREPTR = PTR Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM Step 8: SET PREPTR = PTR Step 9: SET PTR = PTR->NEXT [END OF LOOP] Step 10: PREPTR->NEXT = NEW_NODE Step 11: SET NEW_NODE->NEXT = PTR Step 12: EXIT **Figure 6.18** Algorithm to insert a new node before a node that has value NUM
Singly Linked list		7. Deletion of first node	 Step 1: IF START = NULL Write UNDERFLOW Go to Step 5 [END OF IF] Step 2: SET PTR = START Step 3: SET START = START->NEXT Step 4: FREE PTR Step 5: EXIT **Figure 6.21** Algorithm to delete the first node
Singly Linked list		8. Deletion of last node	 Step 1: IF START = NULL Write UNDERFLOW Go to Step 8 [END OF IF] Step 2: SET PTR = START Step 3: Repeat Steps 4 and 5 while PTR->NEXT != NULL Step 4: SET PREPTR = PTR Step 5: SET PTR = PTR->NEXT [END OF LOOP] Step 6: SET PREPTR->NEXT = NULL Step 7: FREE PTR Step 8: EXIT **Figure 6.23** Algorithm to delete the last node
Singly Linked list		9. Deletion of a node after a given node	

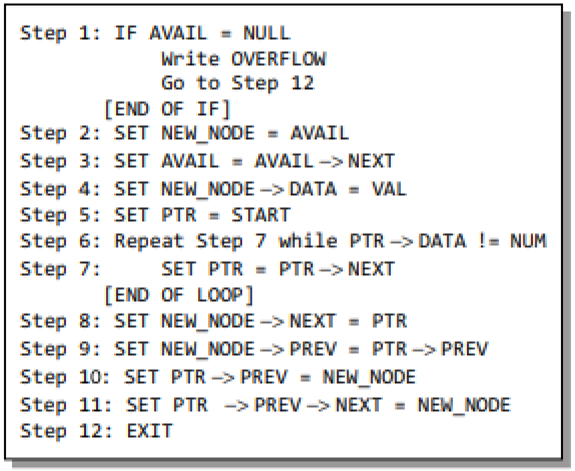
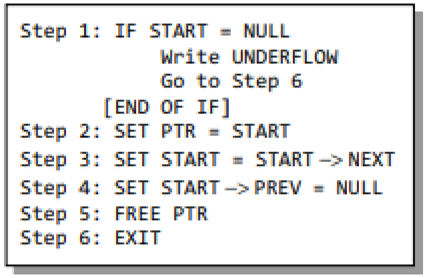
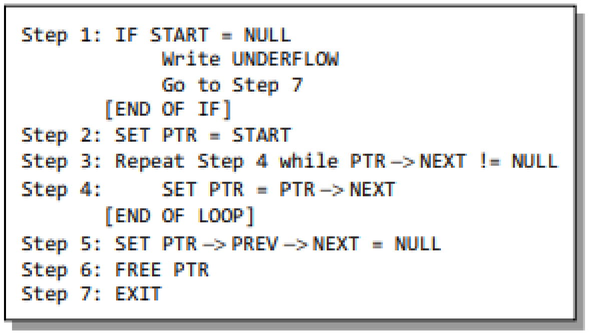
			 Step 1: IF START = NULL Write UNDERFLOW Go to Step 10 [END OF IF] Step 2: SET PTR = START Step 3: SET PREPTR = PTR Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM Step 5: SET PREPTR = PTR Step 6: SET PTR = PTR -> NEXT [END OF LOOP] Step 7: SET TEMP = PTR Step 8: SET PREPTR -> NEXT = PTR -> NEXT Step 9: FREE TEMP Step 10: EXIT **Figure 6.25** Algorithm to delete the node after a given node
Circular Linked list		1. Insertion of a node in the beginning	 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 11 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET PTR = START Step 6: Repeat Step 7 while PTR -> NEXT != START Step 7: PTR = PTR -> NEXT [END OF LOOP] Step 8: SET NEW_NODE -> NEXT = START Step 9: SET PTR -> NEXT = NEW_NODE Step 10: SET START = NEW_NODE Step 11: EXIT **Figure 6.30** Algorithm to insert a new node at the beginning
Circular Linked list		2. Insertion of a node at the end	 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 10 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET NEW_NODE -> NEXT = START Step 6: SET PTR = START Step 7: Repeat Step 8 while PTR -> NEXT != START Step 8: SET PTR = PTR -> NEXT [END OF LOOP] Step 9: SET PTR -> NEXT = NEW_NODE Step 10: EXIT **Figure 6.32** Algorithm to insert a new node at the end
Circular		3. Deletion	

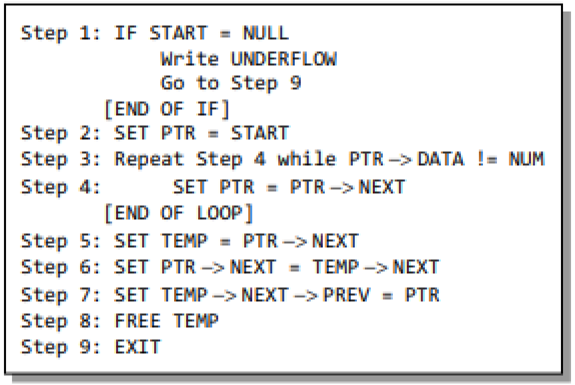
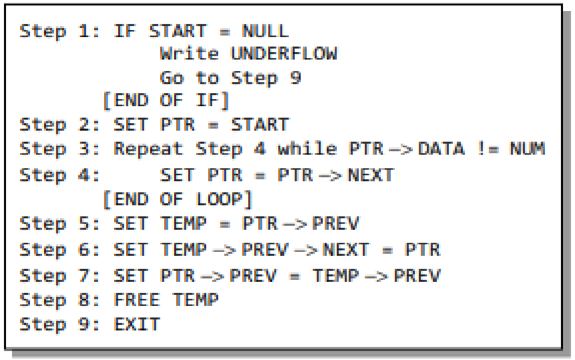
Linked list		of a first node	 Step 1: IF START = NULL Write UNDERFLOW Go to Step 8 [END OF IF] Step 2: SET PTR = START Step 3: Repeat Step 4 while PTR -> NEXT != START Step 4: SET PTR = PTR -> NEXT [END OF LOOP] Step 5: SET PTR -> NEXT = START -> NEXT Step 6: FREE START Step 7: SET START = PTR -> NEXT Step 8: EXIT **Figure 6.34** Algorithm to delete the first node
Circular Linked list		4. Deletion of a last node	 Step 1: IF START = NULL Write UNDERFLOW Go to Step 8 [END OF IF] Step 2: SET PTR = START Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != START Step 4: SET PREPTR = PTR Step 5: SET PTR = PTR -> NEXT [END OF LOOP] Step 6: SET PREPTR -> NEXT = START Step 7: FREE PTR Step 8: EXIT **Figure 6.36** Algorithm to delete the last node

DOUBLY LINKED LISTS	struct node
	{
	struct node *prev;
	int data;
	struct node *next;
	};
		1. Insertion of a new node at the beginning	
			Step 1: IF AVAIL = NULL
			Write OVERFLOW
			Go to Step 9
			[END OF IF]
			Step 2: SET NEW_NODE = AVAIL
			Step 3: SET AVAIL = AVAIL -> NEXT
			Step 4: SET NEW_NODE -> DATA = VAL
			Step 5: SET NEW_NODE -> PREV = NULL
			Step 6: SET NEW_NODE -> NEXT = START
			Step 7: SET START -> PREV = NEW_NODE
			Step 8: SET START = NEW_NODE
			Step 9: EXIT
			Figure 6.40 Algorithm to insert a new node at the beginning

DOUBLY LINKED LISTS		2. Insertion of a new node at the end	

			 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 11 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET NEW_NODE -> NEXT = NULL Step 6: SET PTR = START Step 7: Repeat Step 8 while PTR -> NEXT != NULL Step 8: SET PTR = PTR -> NEXT [END OF LOOP] Step 9: SET PTR -> NEXT = NEW_NODE Step 10: SET NEW_NODE -> PREV = PTR Step 11: EXIT **Figure 6.42** Algorithm to insert a new node at the end
DOUBLY LINKED LISTS		3. Insert the node after a given node	 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 12 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET PTR = START Step 6: Repeat Step 7 while PTR -> DATA != NUM Step 7: SET PTR = PTR -> NEXT [END OF LOOP] Step 8: SET NEW_NODE -> NEXT = PTR -> NEXT Step 9: SET NEW_NODE -> PREV = PTR Step 10: SET PTR -> NEXT = NEW_NODE Step 11: SET PTR -> NEXT -> PREV = NEW_NODE Step 12: EXIT **Figure 6.43** Algorithm to insert a new node after a given node
DOUBLY LINKED LISTS		4. Insert the node before the given node	

			 Step 1: IF AVAIL = NULL Write OVERFLOW Go to Step 12 [END OF IF] Step 2: SET NEW_NODE = AVAIL Step 3: SET AVAIL = AVAIL -> NEXT Step 4: SET NEW_NODE -> DATA = VAL Step 5: SET PTR = START Step 6: Repeat Step 7 while PTR -> DATA != NUM Step 7: SET PTR = PTR -> NEXT [END OF LOOP] Step 8: SET NEW_NODE -> NEXT = PTR Step 9: SET NEW_NODE -> PREV = PTR -> PREV Step 10: SET PTR -> PREV = NEW_NODE Step 11: SET PTR -> PREV -> NEXT = NEW_NODE Step 12: EXIT **Figure 6.45** Algorithm to insert a new node before a given node
DOUBLY LINKED LISTS		5. Deletion of a first node	 Step 1: IF START = NULL Write UNDERFLOW Go to Step 6 [END OF IF] Step 2: SET PTR = START Step 3: SET START = START -> NEXT Step 4: SET START -> PREV = NULL Step 5: FREE PTR Step 6: EXIT **Figure 6.48** Algorithm to delete the first node
DOUBLY LINKED LISTS		6. Deletion of a last node	 Step 1: IF START = NULL Write UNDERFLOW Go to Step 7 [END OF IF] Step 2: SET PTR = START Step 3: Repeat Step 4 while PTR -> NEXT != NULL Step 4: SET PTR = PTR -> NEXT [END OF LOOP] Step 5: SET PTR -> PREV -> NEXT = NULL Step 6: FREE PTR Step 7: EXIT **Figure 6.50** Algorithm to delete the last node
DOUBLY LINKED LISTS		7. Delete node after a given node	

			 Step 1: IF START = NULL Write UNDERFLOW Go to Step 9 [END OF IF] Step 2: SET PTR = START Step 3: Repeat Step 4 while PTR -> DATA != NUM Step 4: SET PTR = PTR -> NEXT [END OF LOOP] Step 5: SET TEMP = PTR -> NEXT Step 6: SET PTR -> NEXT = TEMP -> NEXT Step 7: SET TEMP -> NEXT -> PREV = PTR Step 8: FREE TEMP Step 9: EXIT **Figure 6.52** Algorithm to delete a node after a given node
DOUBLY LINKED LISTS		8. Delete node before a given node	 Step 1: IF START = NULL Write UNDERFLOW Go to Step 9 [END OF IF] Step 2: SET PTR = START Step 3: Repeat Step 4 while PTR -> DATA != NUM Step 4: SET PTR = PTR -> NEXT [END OF LOOP] Step 5: SET TEMP = PTR -> PREV Step 6: SET TEMP -> PREV -> NEXT = PTR Step 7: SET PTR -> PREV = TEMP -> PREV Step 8: FREE TEMP Step 9: EXIT **Figure 6.54** Algorithm to delete a node before a given node