1. Implement a function that takes two linked lists as input and returns their intersection. Suppose here the two lists are sorted.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node\* next;  };  struct Node\* createNode(int data) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  if (newNode == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  newNode->data = data;  newNode->next = NULL;  return newNode;  }  void insert(struct Node\*\* head, int data) {  struct Node\* newNode = createNode(data);  if (\*head == NULL) {  \*head = newNode;  } else {  struct Node\* current = \*head;  while (current->next != NULL) {  current = current->next;  }  current->next = newNode;  }  }  struct Node\* getIntersection(struct Node\* list1, struct Node\* list2) {  **//complete code here**  }  void printList(struct Node\* head) {  while (head != NULL) {  printf("%d ", head->data);  head = head->next;  }  printf("\n");  }  void freeList(struct Node\* head) {  struct Node\* current = head;  while (current != NULL) {  struct Node\* temp = current;  current = current->next;  free(temp);  }  }  int main() {  // Create the first sorted linked list: 1->2->4->6  struct Node\* list1 = NULL;  insert(&list1, 1);  insert(&list1, 2);  insert(&list1, 4);  insert(&list1, 6);  // Create the second sorted linked list: 2->4->5->6  struct Node\* list2 = NULL;  insert(&list2, 2);  insert(&list2, 4);  insert(&list2, 5);  insert(&list2, 6);  printf("List 1: ");  printList(list1);  printf("List 2: ");  printList(list2);  struct Node\* intersection = getIntersection(list1, list2);  printf("Intersection: ");  printList(intersection);  // Free the memory allocated for the linked lists  freeList(list1);  freeList(list2);  freeList(intersection);  return 0;  } |

1. Create a function that partitions a linked list around a given value, such that all nodes less than the value come before all nodes greater than or equal to the value.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node\* next;  };  struct Node\* createNode(int data) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  if (newNode == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  newNode->data = data;  newNode->next = NULL;  return newNode;  }  void insert(struct Node\*\* head, int data) {  struct Node\* newNode = createNode(data);  if (\*head == NULL) {  \*head = newNode;  } else {  struct Node\* current = \*head;  while (current->next != NULL) {  current = current->next;  }  current->next = newNode;  }  }  void partitionLinkedList(struct Node\*\* head, int pivot) {  **//complete code here**  }  void printList(struct Node\* head) {  while (head != NULL) {  printf("%d ", head->data);  head = head->next;  }  printf("\n");  }  void freeList(struct Node\* head) {  struct Node\* current = head;  while (current != NULL) {  struct Node\* temp = current;  current = current->next;  free(temp);  }  }  int main() {  struct Node\* head = NULL;  // Insert nodes into the linked list  insert(&head, 6);  insert(&head, 2);  insert(&head, 8);  insert(&head, 4);  insert(&head, 1);  insert(&head, 9);  printf("Original List: ");  printList(head);  int pivot = 5;  partitionLinkedList(&head, pivot);  printf("Partitioned List: ");  printList(head);  // Free the memory allocated for the linked list  freeList(head);  return 0;  } |

1. Write a function to detect if a linked list has a cycle. If it does, find the node at the beginning of the cycle.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  struct Node {  int data;  struct Node\* next;  };  struct Node\* createNode(int data) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  if (newNode == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  newNode->data = data;  newNode->next = NULL;  return newNode;  }  void insert(struct Node\*\* head, int data) {  struct Node\* newNode = createNode(data);  if (\*head == NULL) {  \*head = newNode;  } else {  struct Node\* current = \*head;  while (current->next != NULL) {  current = current->next;  }  current->next = newNode;  }  }  bool hasCycle(struct Node\* head) {  **//complete code here**  }  struct Node\* detectCycle(struct Node\* head) {  **//complete code here**  }  void printList(struct Node\* head) {  while (head != NULL) {  printf("%d ", head->data);  head = head->next;  }  printf("\n");  } |

1. Implement a function to rotate a linked list to the right by k places, where k is a non-negative integer.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node\* next;  };  struct Node\* createNode(int data) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  if (newNode == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  newNode->data = data;  newNode->next = NULL;  return newNode;  }  void insert(struct Node\*\* head, int data) {  struct Node\* newNode = createNode(data);  if (\*head == NULL) {  \*head = newNode;  } else {  struct Node\* current = \*head;  while (current->next != NULL) {  current = current->next;  }  current->next = newNode;  }  }  int getLength(struct Node\* head) {  int length = 0;  struct Node\* current = head;  while (current != NULL) {  length++;  current = current->next;  }  return length;  }  struct Node\* rotateRight(struct Node\* head, int k) {  **//complete code here**  }  void printList(struct Node\* head) {  while (head != NULL) {  printf("%d ", head->data);  head = head->next;  }  printf("\n");  }  void freeList(struct Node\* head) {  struct Node\* current = head;  while (current != NULL) {  struct Node\* temp = current;  current = current->next;  free(temp);  }  }  int main() {  struct Node\* head = NULL;  // Insert nodes into the linked list  insert(&head, 1);  insert(&head, 2);  insert(&head, 3);  insert(&head, 4);  insert(&head, 5);  printf("Original List: ");  printList(head);  int k = 2; // Rotate the list to the right by 2 places  struct Node\* rotatedList = rotateRight(head, k);  printf("Rotated List: ");  printList(rotatedList);  // Free the memory allocated for the linked list  freeList(rotatedList);  return 0;  } |

1. Implement a function to add two numbers represented as linked lists, where each node contains a digit in the reverse order. However, this time, the digits are stored in the forward order.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node\* next;  };  struct Node\* createNode(int data) {  struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));  if (newNode == NULL) {  printf("Memory allocation failed.\n");  exit(1);  }  newNode->data = data;  newNode->next = NULL;  return newNode;  }  void insert(struct Node\*\* head, int data) {  struct Node\* newNode = createNode(data);  if (\*head == NULL) {  \*head = newNode;  } else {  struct Node\* current = \*head;  while (current->next != NULL) {  current = current->next;  }  current->next = newNode;  }  }  struct Node\* addNumbers(struct Node\* num1, struct Node\* num2) {    **//complete code here**  }  void printList(struct Node\* head) {  while (head != NULL) {  printf("%d ", head->data);  head = head->next;  }  printf("\n");  }  void freeList(struct Node\* head) {  struct Node\* current = head;  while (current != NULL) {  struct Node\* temp = current;  current = current->next;  free(temp);  }  }  int main() {  struct Node\* num1 = NULL;  struct Node\* num2 = NULL;  // Insert digits into the first number (753)  insert(&num1, 7);  insert(&num1, 5);  insert(&num1, 3);  // Insert digits into the second number (248)  insert(&num2, 2);  insert(&num2, 4);  insert(&num2, 8);  printf("Number 1: ");  printList(num1);  printf("Number 2: ");  printList(num2);  struct Node\* sum = addNumbers(num1, num2);  printf("Sum: ");  printList(sum);  // Free the memory allocated for the linked lists  freeList(num1);  freeList(num2);  freeList(sum);  return 0;  } |