Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q1. Write a C program to Insert and Delete elements form a Queue using link list ,each node should have the following inforamaion about a product Product_Id(char) , Product_Name(string) , Total_sale(integer),Product_Grade(Char)

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct product {
char Product_Id;
char Product_Name[100];
int Total_sale;
char Product_Grade;
struct product* next;
};

struct Queue {
struct product* front;
struct product* rear;
};

struct product* createProduct(char id, const char* name, int totalSale, char grade) {
struct product* temp = (struct product*)malloc(sizeof(struct product));
if (temp == NULL) {
printf("Memory not allocated !\n");
exit(1);
}
temp->Product_Id = id;
strcpy(temp->Product_Name, name);
temp->Total_sale = totalSale;
temp->Product_Grade = grade;
temp->next = NULL;
return temp;
}

struct Queue* createQueue() {
struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
if (queue == NULL) {
printf("Memory not allocated !\n");
exit(1);
}
queue->front = NULL;
queue->rear = NULL;
return queue;
}
```

```
int isEmpty(struct Queue* queue) {
return (queue->front == NULL);
}

void insert(struct Queue* queue, struct product* product) {
if (isEmpty(queue)) {
queue->front = product;
queue->rear = product;
} else {
queue->rear->next = product;
queue->rear = product;
}
printf("Product inserted\n");
}

struct product* removeProduct(struct Queue* queue) {
if (isEmpty(queue)) {
printf("Queue is empty\n");
return NULL;
}
struct product* temp = queue->front;
queue->front = queue->front->next;
if (queue->front == NULL) {
queue->rear = NULL;
}
printf("Product removed\n");
return temp;
}

void displayProduct(struct product* product) {
printf("\n");
printf("Product ID: %c\n", product->Product_Id);
printf("Product Name: %s\n", product->Product_Name);
printf("Total Sale: %d\n", product->Total_sale);
printf("Product Grade: %c\n", product->Product_Grade);
}

void displayQueue(struct Queue* queue) {
if (isEmpty(queue)) {
printf("Queue is empty.\n");
return;
}

printf("Products in the Queue:\n");
struct product* current = queue->front;
while (current != NULL) {
displayProduct(current);
current = current->next;
}
```

```
}

void destroyQueue(struct Queue* queue) {
struct product* current = queue->front;
struct product* next;
while (current != NULL) {
next = current->next;
free(current);
current = next;
}
free(queue);
}

int main() {
struct Queue* q = createQueue();
int choice;
char id, name[50], grade;
int totalSale;
do {
printf("\n<--- Menu --->\n");
printf("1. Insert\n");
printf("2. Remove\n");
printf("3. Display\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter Product ID: ");
scanf(" %c", &id);
printf("Enter Product Name: ");
scanf("%s", name);
printf("Enter Total Sale: ");
scanf("%d", &totalSale);
printf("Enter Product Grade: ");
scanf(" %c", &grade);
insert(q, createProduct(id, name, totalSale, grade));
break;
case 2:
removeProduct(q);
break;
case 3:
displayQueue(q);
break;
case 4:
printf("Exiting the program.\n");
break;
default:
printf("Invalid choice ! \n");
}
```

```
} while (choice != 4);
destroyQueue(q);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Insert
2. Remove
3. Display
4. Exit
Enter your choice: 1
Enter Product ID: 1
Enter Product Name: Truke
Enter Total Sale: 20000
Enter Product Grade: A
Product inserted

<--- Menu --->
1. Insert
2. Remove
3. Display
4. Exit
Enter your choice: 1
Enter Product ID: 2
Enter Product Name: Boat
Enter Total Sale: 30000
Enter Product Grade: B
Product inserted

<--- Menu --->
1. Insert
2. Remove
3. Display
4. Exit
Enter your choice: 3
Products in the Queue:

Product ID: 1
Product Name: Truke
Total Sale: 20000
Product Grade: A

Product ID: 2
Product Name: Boat
Total Sale: 30000
Product Grade: B

<--- Menu --->
```

1. Insert
2. Remove
3. Display
4. Exit
Enter your choice: 2
Product removed

<--- Menu --->
1. Insert
2. Remove
3. Display
4. Exit
Enter your choice: 3
Products in the Queue:

Product ID: 2
Product Name: Boat
Total Sale: 30000
Product Grade: B

<--- Menu --->
1. Insert
2. Remove
3. Display
4. Exit
Enter your choice: 2
Product removed

<--- Menu --->
1. Insert
2. Remove
3. Display
4. Exit
Enter your choice: 3
Queue is empty.

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q2. Let A and B be two structures of type Linked List. Write a ‘C ’ program to create a new Linked List ‘S’ that contains elements alternately from A and B beginning with the first element of A. If you run out of elements in one of the lists, then append the remaining.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int data;
struct Node* next;
};

void append(struct Node** head, int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;
if (*head == NULL) {
*head = newNode;
}
else {
struct Node* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
}
}




struct Node* mergeAlternate(struct Node* a, struct Node* b) {
struct Node* result = NULL;
while (a != NULL || b != NULL) {
if (a != NULL) {
append(&result, a->data);
a = a->next;
}
if (b != NULL) {
append(&result, b->data);
b = b->next;
}
}
return result;
}
```

```
void display(struct Node* head) {
while (head != NULL) {
printf("%d ", head->data);
head = head->next;
}
printf("\n");
}

void freeList(struct Node* head) {
struct Node* temp;
while (head != NULL) {
temp = head;
head = head->next;
free(temp);
}
}
int main() {
struct Node* A = NULL;
struct Node* B = NULL;
struct Node* S = NULL;
int choice;
do {
printf("\n<--- Menu --->\n");
printf("1. Create LL A\n");
printf("2. Create LL B\n");
printf("3. Merge LL A & B \n");
printf("4. Display LL S\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter elements for Linked List A (enter -1 to stop):\n");
int element;
while (1) {
scanf("%d", &element);
if (element == -1) break;
append(&A, element);
}
break;
case 2:
printf("Enter elements for Linked List B (enter -1 to stop):\n");
while (1) {
scanf("%d", &element);
if (element == -1) break;
append(&B, element);
}
break;
case 3:
```

```
S = mergeAlternate(A, B);
printf("Merged alternately.\n");
break;
case 4:
printf("Merged List S: ");
display(S);
break;
case 5:
break;
default:
printf("Invalid !\n");
}
} while (choice != 5);
freeList(A);
freeList(B);
freeList(S);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Create LL A
2. Create LL B
3. Merge LL A & B
4. Display LL S
5. Exit
Enter your choice: 1
Enter elements for Linked List A (enter -1 to stop):
2
4
6
8
-1

<--- Menu --->
1. Create LL A
2. Create LL B
3. Merge LL A & B
4. Display LL S
5. Exit
Enter your choice: 2
Enter elements for Linked List B (enter -1 to stop):
1
3
5
7
-1
```

<--- Menu --->
1. Create LL A
2. Create LL B
3. Merge LL A & B
4. Display LL S
5. Exit
Enter your choice: 3
Merged alternately.

<--- Menu --->
1. Create LL A
2. Create LL B
3. Merge LL A & B
4. Display LL S
5. Exit
Enter your choice: 4
Merged List S: 2 1 4 3 6 5 8 7

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q3.Write a C program to create a single linked list then input a value V, partition it such that all nodes less than V come before nodes greater than or equal to V.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int data;
struct Node* next;
};

void append(struct Node** head, int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;
if (*head == NULL) {
*head = newNode;
}
else {
struct Node* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
}
}


struct Node* partition(struct Node* head, int V) {
struct Node* temp1 = NULL;
struct Node* temp2 = NULL;
while (head != NULL) {
if (head->data < V) {
append(&temp1, head->data);
}
else {
append(&temp2, head->data);
}
head = head->next;
}
if (temp1 == NULL) {
return temp2;
}
else {
```

```
struct Node* temp = temp1;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = temp2;
return temp1;
}
}

void printList(struct Node* head) {
while (head != NULL) {
printf("%d ", head->data);
head = head->next;
}
printf("\n");
}
void freeList(struct Node* head) {
struct Node* temp;
while (head != NULL) {
temp = head;
head = head->next;
free(temp);
}
}

int main() {
struct Node* list = NULL;
int choice;
do {
printf("\n<--- Menu --->\n");
printf("1. Create Linked List\n");
printf("2. Partition Linked List\n");
printf("3. Display Partitioned List\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter elements for the Linked List (enter -1 to stop):\n");
int element;
while (1) {
scanf("%d", &element);
if (element == -1) break;
append(&list, element);
}
break;
case 2:
if (list == NULL) {
printf("Linked List is empty\n");
} else {
```

```
int V;
printf("Enter the value V for partition: ");
scanf("%d", &V);
list = partition(list, V);
printf("Linked List partitioned for %d.\n", V);
}
break;
case 3:
if (list == NULL) {
printf("Linked List is empty.\n");
} else {
printf("Partitioned List: ");
printList(list);
}
break;
case 4:
break;
default:
printf("Invalid choice !\n");
}
} while (choice != 4);
freeList(list);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Create Linked List
2. Partition Linked List
3. Display Partitioned List
4. Exit
Enter your choice: 1
Enter elements for the Linked List (enter -1 to stop):
15
10
11
49
50
-1

<--- Menu --->
1. Create Linked List
2. Partition Linked List
3. Display Partitioned List
4. Exit
Enter your choice: 2
Enter the value V for partition: 15
Linked List partitioned for 15.
```

<--- Menu --->
1. Create Linked List
2. Partition Linked List
3. Display Partitioned List
4. Exit
Enter your choice: 3
Partitioned List: 10 11 15 49 50

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q4. Write a C program to create two single linked lists, and then write another function to subtract two numbers represented as linked list.
List1->; 8->9->7->NULL (First Number: 897)
List2->: 1->4->5->NULL (Second Number: 145)
Output->:752

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int data;
struct Node* next;
};

void append(struct Node** head, int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;
if (*head == NULL) {
*head = newNode;
}
else {
struct Node* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
}
}


void display(struct Node* head) {
while (head != NULL) {
printf("%d", head->data);
if (head->next != NULL) {
printf("->");
}
head = head->next;
}
printf("\n");
}

struct Node* subtract(struct Node* list1, struct Node* list2) {
struct Node* result = NULL;
struct Node* temp1 = list1;
```

```
struct Node* temp2 = list2;
int borrow = 0;
while (temp1 != NULL || temp2 != NULL) {
int value1 = (temp1 != NULL) ? temp1->data : 0;
int value2 = (temp2 != NULL) ? temp2->data : 0;
int diff = value1 - value2 - borrow;
borrow = 0;
if (diff < 0) {
diff += 10;
borrow = 1;
}
append(&result, diff);
if (temp1 != NULL) {
temp1 = temp1->next;
}
if (temp2 != NULL) {
temp2 = temp2->next;
}
}
return result;
}

void freeList(struct Node* head) {
struct Node* temp;
while (head != NULL) {
temp = head;
head = head->next;
free(temp);
}
}

int main() {
struct Node* list1 = NULL;
struct Node* list2 = NULL;
struct Node* result = NULL;
int choice;
do {
printf("\n<--- Menu --->\n");
printf("1. Create LL 1\n");
printf("2. Create LL 2\n");
printf("3. Subtract Linked List\n");
printf("4. Display\n");
printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter elements for Linked List 1 (enter -1 to stop):\n");
int element;
while (1) {
```

```
scanf("%d", &element);
if (element == -1) break;
append(&list1, element);
}
break;
case 2:
printf("Enter elements for Linked List 2 (enter -1 to stop):\n");
while (1) {
scanf("%d", &element);
if (element == -1) break;
append(&list2, element);
}
break;
case 3:
if (list1 == NULL || list2 == NULL) {
printf("Error ! \n");
} else {
result = subtract(list1, list2);
printf("Lists subtracted successfully.\n");
}
break;
case 4:
if (result == NULL) {
printf("Result is empty.\n");
} else {
printf("Result: ");
display(result);
}
break;
case 5:
break;
default:
printf("Invalid !\n");
}
} while (choice != 5);
freeList(list1);
freeList(list2);
freeList(result);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Create LL 1
2. Create LL 2
3. Subtract Linked List
4. Display
5. Exit
Enter your choice: 1
```

```
Enter elements for Linked List 1 (enter -1 to stop):
15
23
40
55
60
-1

<--- Menu --->
1. Create LL 1
2. Create LL 2
3. Subtract Linked List
4. Display
5. Exit
Enter your choice: 2
Enter elements for Linked List 2 (enter -1 to stop):
10
8
20
34
57
-1

<--- Menu --->
1. Create LL 1
2. Create LL 2
3. Subtract Linked List
4. Display
5. Exit
Enter your choice: 3
Lists subtracted successfully.

<--- Menu --->
1. Create LL 1
2. Create LL 2
3. Subtract Linked List
4. Display
5. Exit
Enter your choice: 4
Result: 5->15->20->21->3
```

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q5. Write a C program to craeate a single linked list , like L0 -> L1 -> ... -> Ln-1 -> Ln. Write another C fucntion to rearrange the nodes in the list so that the new formed list is : L0 -> Ln -> L1 -> Ln-1 -> L2-> Ln-2 .You are required to do this in place without altering the nodes' values.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int data;
struct Node* next;
};

void append(struct Node** head, int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;

if (*head == NULL) {
*head = newNode;
} else {
struct Node* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
}
}



void printList(struct Node* head) {
while (head != NULL) {
printf("%d", head->data);
if (head->next != NULL) {
printf(" -> ");
}
head = head->next;
}
printf("\n");
}

void rearrangeList(struct Node** head) {
if (*head == NULL || (*head)->next == NULL) {
```

```
return;
}
struct Node* slow = *head;
struct Node* fast = *head;
while (fast->next != NULL && fast->next->next != NULL) {
slow = slow->next;
fast = fast->next->next;
}
struct Node* prev = NULL;
struct Node* current = slow->next;
struct Node* next;
while (current != NULL) {
next = current->next;
current->next = prev;
prev = current;
current = next;
}
slow->next = NULL;
struct Node* first = *head;
struct Node* second = prev;
while (second != NULL) {
struct Node* nextFirst = first->next;
struct Node* nextSecond = second->next;
first->next = second;
second->next = nextFirst;
first = nextFirst;
second = nextSecond;
}
}

void freeList(struct Node* head) {
struct Node* temp;
while (head != NULL) {
temp = head;
head = head->next;
free(temp);
}
}


int main() {
struct Node* list = NULL;
int choice;
do {
printf("\n<--- Menu --->\n");
printf("1. Create Linked List\n");
printf("2. Display Original LL\n");
printf("3. Rearrange\n");
printf("4. Display Rearranged LL\n");
printf("5. Exit\n");
```

```
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1: {
printf("Enter elements for the Linked List (enter -1 to stop):\n");
int element;
while (1) {
scanf("%d", &element);
if (element == -1) break;
append(&list, element);
}
break;
}
case 2:
printf("Original List: ");
printList(list);
break;
case 3:
if (list == NULL) {
printf(" List is empty !\n");
} else {
rearrangeList(&list);
printf("List rearranged successfully.\n");
}
break;
case 4:
if (list == NULL) {
printf("Linked List is empty.\n");
} else {
printf("Rearranged List: ");
printList(list);
}
break;
case 5:
break;
default:
printf("Invalid ! \n");
}
} while (choice != 5);
freeList(list);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Create Linked List
2. Display Original LL
3. Rearrange
4. Display Rearranged LL
5. Exit
Enter your choice: 1
Enter elements for the Linked List (enter -1 to stop):
13
55
67
34
28
97
-1

<--- Menu --->
1. Create Linked List
2. Display Original LL
3. Rearrange
4. Display Rearranged LL
5. Exit
Enter your choice: 2
Original List: 13 -> 55 -> 67 -> 34 -> 28 -> 97

<--- Menu --->
1. Create Linked List
2. Display Original LL
3. Rearrange
4. Display Rearranged LL
5. Exit
Enter your choice: 3
List rearranged successfully.

<--- Menu --->
1. Create Linked List
2. Display Original LL
3. Rearrange
4. Display Rearranged LL
5. Exit
Enter your choice: 4
Rearranged List: 13 -> 97 -> 55 -> 28 -> 67 -> 34
```

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q6. Write a C program to create two link lists positive and negative from Original linked list, so that positive linked list contains all positive elements and negative linked list contains negative elements. Positive and negative linked lists should use the node of existing original linked list.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int data;
struct Node* next;
};

void append(struct Node** head, int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;

if (*head == NULL) {
*head = newNode;
} else {
struct Node* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
}
}

void display(struct Node* head) {
while (head != NULL) {
printf("%d", head->data);
if (head->next != NULL) {
printf(" -> ");
}
head = head->next;
}
printf("\n");
}

void split(struct Node* original, struct Node** positive, struct Node** negative) {
struct Node* current = original;
while (current != NULL) {
if (current->data >= 0) {
append(positive, current->data);
```

```
}
else {
append(negative, current->data);
}
current = current->next;
}
}


void freeList(struct Node* head) {
struct Node* temp;
while (head != NULL) {
temp = head;
head = head->next;
free(temp);
}
}

int main() {
struct Node* original = NULL;
struct Node* positive = NULL;
struct Node* negative = NULL;
int choice;
do {
printf("\n<--- Menu --->\n");
printf("1. Create Original Linked List\n");
printf("2. Display Original Linked List\n");
printf("3. Split Linked List (+)ve & (-)ve\n");
printf("4. Display (+)ve Linked List\n");
printf("5. Display (-)ve Linked List\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1: {
printf("Enter elements for the Original Linked List (enter -0 to stop):\n");
int element;
while (1) {
scanf("%d", &element);
if (element == -0) break;
append(&original, element);
}
break;
}
case 2:
printf("Original Linked List: ");
display(original);
break;
case 3:
if (original == NULL) {
```

```
printf("Original Linked List is empty\n");
} else {
split(original, &positive, &negative);
printf("Linked List split into (+)ve & (-)ve successfully.\n");
}
break;
case 4:
printf("Positive Linked List: ");
display(positive);
break;
case 5:
printf("Negative Linked List: ");
display(negative);
break;
case 6:
break;
default:
printf("Invalid choice !\n");
}
} while (choice != 6);

freeList(original);
freeList(positive);
freeList(negative);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Create Original Linked List
2. Display Original Linked List
3. Split Linked List (+)ve & (-)ve
4. Display (+)ve Linked List
5. Display (-)ve Linked List
6. Exit
Enter your choice: 1
Enter elements for the Original Linked List (enter -0 to stop):
10
-20
30
-40
50
-60
70
-80
90
-100
-0
```

<--- Menu --->
1. Create Original Linked List
2. Display Original Linked List
3. Split Linked List (+)ve & (-)ve
4. Display (+)ve Linked List
5. Display (-)ve Linked List
6. Exit
Enter your choice: 2
Original Linked List: 10 -> -20 -> 30 -> -40 -> 50 -> -60 -> 70 -> -80 -> 90 -> -100

<--- Menu --->
1. Create Original Linked List
2. Display Original Linked List
3. Split Linked List (+)ve & (-)ve
4. Display (+)ve Linked List
5. Display (-)ve Linked List
6. Exit
Enter your choice: 3
Linked List split into (+)ve & (-)ve successfully.

<--- Menu --->
1. Create Original Linked List
2. Display Original Linked List
3. Split Linked List (+)ve & (-)ve
4. Display (+)ve Linked List
5. Display (-)ve Linked List
6. Exit
Enter your choice: 4
Positive Linked List: 10 -> 30 -> 50 -> 70 -> 90

<--- Menu --->
1. Create Original Linked List
2. Display Original Linked List
3. Split Linked List (+)ve & (-)ve
4. Display (+)ve Linked List
5. Display (-)ve Linked List
6. Exit
Enter your choice: 5
Negative Linked List: -20 -> -40 -> -60 -> -80 -> -100

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q7. W.A.P. to create a binary search tree and perform following operations:
1) Search a particular key.
2) Delete a node from the tree.
3) Find total number of leaf nodes
4) Find height of a binary search tree
5) Count total numbers of nodes from right hand side of root node
6) Kth largest element without doing any modification in Binary Search Tree.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node* left;
   struct Node* right;
};

struct Node* createNode(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->left = newNode->right = NULL;
   return newNode;
}

struct Node* insert(struct Node* root, int value) {
   if (root == NULL) {
      return createNode(value);
   }

   if (value < root->data) {
      root->left = insert(root->left, value);
   } else if (value > root->data) {
      root->right = insert(root->right, value);
   }

   return root;
}

struct Node* search(struct Node* root, int value) {
   if (root == NULL || root->data == value) {
      return root;
   }

   if (value < root->data) {
      return search(root->left, value);
```

```
    }

    return search(root->right, value);
}

struct Node* minValueNode(struct Node* node) {
    struct Node* current = node;

    while (current->left != NULL) {
        current = current->left;
    }

    return current;
}

struct Node* deleteNode(struct Node* root, int value) {
    if (root == NULL) {
        return root;
    }

    if (value < root->data) {
        root->left = deleteNode(root->left, value);
    } else if (value > root->data) {
        root->right = deleteNode(root->right, value);
    } else {
        if (root->left == NULL) {
            struct Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct Node* temp = root->left;
            free(root);
            return temp;
        }

        struct Node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}

int countLeaves(struct Node* root) {
    if (root == NULL) {
        return 0;
    }

    if (root->left == NULL && root->right == NULL) {
        return 1;
```

```
    }

    return countLeaves(root->left) + countLeaves(root->right);
}

int max(int a, int b) {
    return (a > b) ? a : b;
}

int height(struct Node* root) {
    if (root == NULL) {
        return 0;
    }

    return 1 + max(height(root->left), height(root->right));
}

void countNodesRight(struct Node* root, int level, int k, int* count) {
    if (root == NULL) {
        return;
    }

    if (level == k) {
        (*count)++;
        return;
    }

    countNodesRight(root->right, level + 1, k, count);
    countNodesRight(root->left, level + 1, k, count);
}

void kLargest(struct Node* root, int k, int* count, int* found, int* result) {
    if (root == NULL || (*count) >= k || *found) {
        return;
    }

    kLargest(root->right, k, count, found, result);

    (*count)++;

    if (*count == k) {
        *found = 1;
        *result = root->data;
        return;
    }

    kLargest(root->left, k, count, found, result);
}

void kthLargest(struct Node* root, int k) {
```

```
    int count = 0;
    int found = 0;
    int result;

    kLargest(root, k, &count, &found, &result);

    if (found) {
        printf("%d-th largest element: %d\n", k, result);
    } else {
        printf("Invalid value of k or tree does not have enough elements.\n");
    }
}

void freeTree(struct Node* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}

int main() {
    struct Node* root = NULL;
    int choice, value;

    do {
        printf("\n<--- Menu --->\n");
        printf("1. Insert\n");
        printf("2. Search\n");
        printf("3. Delete\n");
        printf("4. Count Leaf Nodes\n");
        printf("5. Calculate Height\n");
        printf("6. Count Nodes from Right\n");
        printf("7. Kth Largest Element\n");
        printf("8. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data to insert: ");
                scanf("%d", &value);
                root = insert(root, value);
                break;

            case 2:
                printf("Enter value to search: ");
                scanf("%d", &value);
                if (search(root, value) != NULL) {
                    printf("%d found in the tree.\n", value);
```

```
            } else {
                printf("%d not found in the tree.\n", value);
            }
            break;

        case 3:
            printf("Enter value to delete: ");
            scanf("%d", &value);
            root = deleteNode(root, value);
            break;

        case 4:
            printf("Total number of leaf nodes: %d\n", countLeaves(root));
            break;

        case 5:
            printf("Height of the BST : %d\n", height(root));
            break;

        case 6: {
            int level, count = 0;
            printf("Enter level from right: ");
            scanf("%d", &level);
            countNodesRight(root, 1, level, &count);
            printf("Total nodes from right at level %d: %d\n", level, count);
            break;
        }

        case 7:
            printf("Enter value of k for kth largest element: ");
            scanf("%d", &value);
            kthLargest(root, value);
            break;

        case 8:
            break;

        default:
            printf("Invalid choice !\n");
      }
   } while (choice != 8);
   freeTree(root);

   return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Insert
```

2. Search
3. Delete
4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 1
Enter data to insert: 13

<--- Menu --->
1. Insert
2. Search
3. Delete
4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 1
Enter data to insert: 24

<--- Menu --->
1. Insert
2. Search
3. Delete
4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 1
Enter data to insert: 35

<--- Menu --->
1. Insert
2. Search
3. Delete
4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 1
Enter data to insert: 46

<--- Menu --->
1. Insert
2. Search
3. Delete

4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 2
Enter value to search: 35
35 found in the tree.

<--- Menu --->
1. Insert
2. Search
3. Delete
4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 4
Total number of leaf nodes: 1

<--- Menu --->
1. Insert
2. Search
3. Delete
4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 5
Height of the BST : 4

<--- Menu --->
1. Insert
2. Search
3. Delete
4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 6
Enter level from right: 1
Total nodes from right at level 1: 1

<--- Menu --->
1. Insert
2. Search
3. Delete
4. Count Leaf Nodes
5. Calculate Height
6. Count Nodes from Right
7. Kth Largest Element
8. Exit
Enter your choice: 7
Enter value of k for kth largest element: 2
2-th largest element: 35

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q8. Write a program to add of two polynomials of degree n, using linked list
For example :-
p1=anxn+an-1xn-1 + an-2xn-2 + ... a0x0 P2=bnxn+bn-1xn-1 + bn-2xn-2 +..... b0x0
p1 = first polynomial
p2 = second polynomial
Find out p3= p1+p2

SOURCE CODE :-

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
int coff;
int exp;
struct Node* next;
};

struct Node* createNode(int coff, int exp) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->coff = coff;
newNode->exp = exp;
newNode->next = NULL;
return newNode;
}
void insert(struct Node** head, int coff, int exp) {
struct Node* newNode = createNode(coff, exp);
newNode->next = *head;
*head = newNode;
}

void display(struct Node* head) {
struct Node* current = head;
while (current != NULL) {
printf("%dx^%d", current->coff, current->exp);
current = current->next;
if (current != NULL) {
printf(" + ");
}
}
printf("\n");
}

struct Node* addPolynomials(struct Node* p1, struct Node* p2) {
struct Node* result = NULL;
while (p1 != NULL && p2 != NULL) {
if (p1->exp > p2->exp) {
```

```
insert(&result, p1->coff, p1->exp);
p1 = p1->next;
} else if (p1->exp < p2->exp) {
insert(&result, p2->coff, p2->exp);
p2 = p2->next;
} else {
insert(&result, p1->coff + p2->coff, p1->exp);
p1 = p1->next;
p2 = p2->next;
}
}
while (p1 != NULL) {
insert(&result, p1->coff, p1->exp);
p1 = p1->next;
}
while (p2 != NULL) {
insert(&result, p2->coff, p2->exp);
p2 = p2->next;
}
return result;
}

void freeList(struct Node* head) {
struct Node* current = head;
while (current != NULL) {
struct Node* next = current->next;
free(current);
current = next;
}
}

int main() {
struct Node* p1 = NULL;
struct Node* p2 = NULL;
struct Node* result = NULL;
int choice, coff, exp;
do {
printf("\n<--- Menu --->\n");
printf("1. Insert term into p1\n");
printf("2. Insert term into p2\n");
printf("3. Display p1\n");
printf("4. Display p2\n");
printf("5. Add p1 and p2 (p3 = p1 + p2)\n");
printf("6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("Enter coff and exp for p1 term: ");
scanf("%d %d", &coff, &exp);
```

```
insert(&p1, coff, exp);
break;
case 2:
printf("Enter coff and exp for p2 term: ");
scanf("%d %d", &coff, &exp);
insert(&p2, coff, exp);
break;
case 3:
printf("Polynomial p1: ");
display(p1);
break;
case 4:
printf("Polynomial p2: ");
display(p2);
break;
case 5:
result = addPolynomials(p1, p2);
printf("Sum p3 = p1 + p2: ");
display(result);
freeList(result);
break;
case 6:
break;
default:
printf("Invalid choice!\n");
}
} while (choice != 6);
freeList(p1);
freeList(p2);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Insert p1
2. Insert p2
3. Display p1
4. Display p2
5. Add p1 and p2 (p3 = p1 + p2)
6. Exit
Enter your choice: 1
Enter coff and exp for p1 term: 2
3

<--- Menu --->
1. Insert p1
2. Insert p2
3. Display p1
4. Display p2
```

5. Add p1 and p2 (p3 = p1 + p2)
6. Exit
Enter your choice: 1
Enter coff and exp for p1 term: 3
4

<--- Menu --->
1. Insert p1
2. Insert p2
3. Display p1
4. Display p2
5. Add p1 and p2 (p3 = p1 + p2)
6. Exit
Enter your choice: 3
Polynomial p1: 3x^4 + 2x^3

<--- Menu --->
1. Insert p1
2. Insert p2
3. Display p1
4. Display p2
5. Add p1 and p2 (p3 = p1 + p2)
6. Exit
Enter your choice: 2
Enter coff and exp for p2 term: 3
3

<--- Menu --->
1. Insert p1
2. Insert p2
3. Display p1
4. Display p2
5. Add p1 and p2 (p3 = p1 + p2)
6. Exit
Enter your choice: 2
Enter coff and exp for p2 term: 5
4

<--- Menu --->
1. Insert p1
2. Insert p2
3. Display p1
4. Display p2
5. Add p1 and p2 (p3 = p1 + p2)
6. Exit
Enter your choice: 4
Polynomial p2: 5x^4 + 3x^3

<--- Menu --->
1. Insert p1
2. Insert p2
3. Display p1
4. Display p2
5. Add p1 and p2 (p3 = p1 + p2)
6. Exit
Enter your choice: 5
Sum p3 = p1 + p2: 5x^3 + 8x^4

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q9. Write a C program to sort a sequence of characters given by user in an array, using Quick sort technique.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

void swap(char* a, char* b) {
char temp = *a;
*a = *b;
*b = temp;
}

int partition(char arr[], int low, int high) {
char pivot = arr[high];
int i = (low - 1);
for (int j = low; j <= high - 1; j++) {
if (arr[j] < pivot) {
i++;
swap(&arr[i], &arr[j]);
}
}
swap(&arr[i + 1], &arr[high]);
return (i + 1);
}
void quickSort(char arr[], int low, int high) {
if (low < high) {
int pi = partition(arr, low, high);
quickSort(arr, low, pi - 1);
quickSort(arr, pi + 1, high);
}
}

int main() {
int choice, n;
char* arr = NULL;
do {
printf("\n<--- Menu --->\n");
printf("1. Enter characters \n");
printf("2. Quick Sort\n");
printf("3. Display\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
```

```
printf("Enter the number of characters: ");
scanf("%d", &n);
arr = (char*)malloc(n * sizeof(char));
printf("Enter the characters:\n");
for (int i = 0; i < n; i++) {
scanf(" %c", &arr[i]);
}
break;
case 2:
if (arr == NULL) {
printf("Array is not initialized !\n");
} else {
quickSort(arr, 0, n - 1);
printf("Array sorted\n");
}
break;
case 3:
if (arr == NULL) {
printf("Array not initialized !\n");
} else {
printf("Sorted array: ");
for (int i = 0; i < n; i++) {
printf("%c ", arr[i]);
}
printf("\n");
}
break;
case 4:
break;
default:
printf("Invalid choice!\n");
}
} while (choice != 4);
free(arr);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Enter characters
2. Quick Sort
3. Display
4. Exit
Enter your choice: 1
Enter the number of characters: 4
Enter the characters:
```

r
a
j
a

<--- Menu --->
1. Enter characters
2. Quick Sort
3. Display
4. Exit
Enter your choice: 2
Array sorted

<--- Menu --->
1. Enter characters
2. Quick Sort
3. Display
4. Exit
Enter your choice: 3
Sorted array: a a j r

Name : Raja Digvijay Singh
Section : C1
University Roll No : 2219396

Q10. Write a C program to implement BFS.

SOURCE CODE :

```
#include <stdio.h>
#include <stdlib.h>

struct TreeNode {
int data;
struct TreeNode* left;
struct TreeNode* right;
};

struct QueueNode {
struct TreeNode* data;
struct QueueNode* next;
};

struct Queue {
struct QueueNode* front;
struct QueueNode* rear;
};

struct TreeNode* createTreeNode(int value) {
struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
newNode->data = value;
newNode->left = newNode->right = NULL;
return newNode;
}

struct QueueNode* createQueueNode(struct TreeNode* treeNode) {
struct QueueNode* newNode = (struct QueueNode*)malloc(sizeof(struct QueueNode));
newNode->data = treeNode;
newNode->next = NULL;
return newNode;
}

struct Queue* createQueue() {
struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
queue->front = queue->rear = NULL;
return queue;
}

void enqueue(struct Queue* queue, struct TreeNode* treeNode) {
struct QueueNode* newNode = createQueueNode(treeNode);
if (queue->rear == NULL) {
queue->front = queue->rear = newNode;
```

```
} else {
queue->rear->next = newNode;
queue->rear = newNode;
}
}
struct TreeNode* dequeue(struct Queue* queue) {
if (queue->front == NULL) {
return NULL;
}

struct TreeNode* dequeuedNode = queue->front->data;
struct QueueNode* temp = queue->front;
queue->front = queue->front->next;
if (queue->front == NULL) {
queue->rear = NULL;
}
free(temp);
return dequeuedNode;
}

void bfsTraversal(struct TreeNode* root) {
if (root == NULL) {
return;
}

struct Queue* queue = createQueue();
enqueue(queue, root);
while (queue->front != NULL) {
struct TreeNode* currentNode = dequeue(queue);
printf("%d ", currentNode->data);
if (currentNode->left != NULL) {
enqueue(queue, currentNode->left);
}
if (currentNode->right != NULL) {
enqueue(queue, currentNode->right);
}
}
free(queue);
}

void freeTree(struct TreeNode* root) {
if (root != NULL) {
freeTree(root->left);
freeTree(root->right);
free(root);
}
}

struct TreeNode* createTree() {
int value;
```

```
printf("Enter the value : ");
scanf("%d", &value);
struct TreeNode* root = createTreeNode(value);
printf("Left child of %d? (1 for yes, 0 for no): ", value);
int addLeft;
scanf("%d", &addLeft);
if (addLeft) {
root->left = createTree();
}
printf("Right child of %d? (1 for yes, 0 for no): ", value);
int addRight;
scanf("%d", &addRight);
if (addRight) {
root->right = createTree();
}
return root;
}

int main() {
struct TreeNode* root = NULL;
int choice;
do {
printf("\n<--- Menu --->\n");
printf("1. Create tree\n");
printf("2. Perform BFS traversal\n");
printf("3. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
switch (choice) {
case 1:
if (root != NULL) {
freeTree(root);
}
root = createTree();
break;
case 2:
if (root == NULL) {
printf("Tree is not created !\n");
} else {
printf("BFS Traversal: ");
bfsTraversal(root);
printf("\n");
}
break;
case 3:
break;
default:
printf("Invalid choice!\n");
}
} while (choice != 3);
```

```
freeTree(root);
return 0;
}
```

OUTPUT :

```
<--- Menu --->
1. Create tree
2. Perform BFS traversal
3. Exit
Enter your choice: 1
Enter the value : 20
Left child of 20? (1 for yes, 0 for no): 1
Enter the value : 10
Left child of 10? (1 for yes, 0 for no): 1
Enter the value : 5
Left child of 5? (1 for yes, 0 for no): 0
Right child of 5? (1 for yes, 0 for no): 0
Right child of 10? (1 for yes, 0 for no): 1
Enter the value : 15
Left child of 15? (1 for yes, 0 for no): 0
Right child of 15? (1 for yes, 0 for no): 0
Right child of 20? (1 for yes, 0 for no): 1
Enter the value : 30
Left child of 30? (1 for yes, 0 for no): 1
Enter the value : 25
Left child of 25? (1 for yes, 0 for no): 0
Right child of 25? (1 for yes, 0 for no): 0
Right child of 30? (1 for yes, 0 for no): 1
Enter the value : 35
Left child of 35? (1 for yes, 0 for no): 0
Right child of 35? (1 for yes, 0 for no): 0

<--- Menu --->
1. Create tree
2. Perform BFS traversal
3. Exit
Enter your choice: 2
BFS Traversal: 20 10 30 5 15 25 35
```