1. Create two linked list in one linked (1,2,3,4}and in the 2nd linked list will have value(7,8,9).Concatenate both the linked list and display the concatenated linked list.

```
#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node* next;

} Node;

Node* createNode(int data) {

Node* newNode = (Node*)malloc(sizeof(Node));

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void append(Node** head, int data) {

Node* newNode = createNode(data);

if (*head == NULL) {

*head = newNode;

return;

}

Node* temp = *head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;
```

```
}
void concatenate (Node** head1, Node* head2) {
if (*head1 == NULL) {
*head1 head2;
return;
}
Node* temp = *head1;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = head2;
}
void display(Node* head) {
Node* temp = head;
while (temp != NULL) {
printf("%d", temp->data);
temp = temp->next;
}
printf("\n");
}
int main() {
Node* list1 = NULL;
Node* list2 = NULL;
append(&list1, 1);
append(&list1, 2);

append(&list1, 3);

append(&list1, 4);

append(&list2, 7);

append(&list2, 8);

append(&list2, 9);

concatenate(&list1, list2);

display(list1);

return 0;

)
```

2. Problem Statement: Automotive Manufacturing Plant Management System

Objective:

Develop a program to manage an automotive manufacturing plant's operations using a linked list in C programming. The system will allow creation, insertion, deletion, and searching operations for managing assembly lines and their details.

Requirements

Data Representation

1. Node Structure:

Each node in the linked list represents an assembly line.

Fields:

0 lineID (integer): Unique identifier for the assembly line.

lineName (string): Name of the assembly line (e.g., "Chassis Assembly").

capacity (integer): Maximum production capacity of the line per shift.

0 status (string): Current status of the line (e.g., "Active", "Under Maintenance").

0 next (pointer to the next node): Link to the next assembly line in the list.

2. Linked List:

0 The linked list will store a dynamic number of assembly lines, allowing for additions and removals as needed.

Features to Implement

1. Creation:

0 Initialize the linked list with a specified number of assembly lines.

2. Insertion:

0 Add a new assembly line to the list either at the beginning, end, or at a specific position.

3. Deletion:

0 Remove an assembly line from the list by its lineID or position.

4. Searching:

0 Search for an assembly line by lineID or lineName and display its details.

5. Display:

0 Display all assembly lines in the list along with their details.

6. Update Status:

0 Update the status of an assembly line (e.g., from "Active" to "Under Maintenance").

Example Program Flow

1. Menu Options:

Provide a menu-driven interface with the following operations:

0 Create Linked List of Assembly Lines

0 Insert New Assembly Line

Delete Assembly Line

0 Search for Assembly Line

0 Update Assembly Line Status

0 Display All Assembly Lines

2. Sample Input/Output:

Exit

Input:

0 Number of lines: 3

0 Line 1: ID = 101, Name = "Chassis Assembly", Capacity = 50, Status = "Active".

0 Line 2: ID = 102, Name = "Engine Assembly", Capacity = 40, Status = "Under Maintenance".

Output:

```
Assembly Lines:

Line 101: Chassis Assembly, Capacity: 50, Status: Active

Line 102: Engine Assembly, Capacity: 40, Status: Under Maintenance
```

Linked List Node Structure in C

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Structure for a linked list node

typedef struct AssemblyLine {

int lineID;  // Unique line ID

char lineName[50];  // Name of the assembly line

int capacity;  // Production capacity per shift

char status[20];  // Current status of the line

struct AssemblyLine* next; // Pointer to the next node

} AssemblyLine;
```

Operations Implementation

1. Create Linked List

Allocate memory dynamically for AssemblyLine nodes.

Initialize each node with details such as lineID, lineName, capacity, and

status.

2. Insert New Assembly Line

Dynamically allocate a new node and insert it at the desired position in the list.

3. Delete Assembly Line

Locate the node to delete by lineID or position and adjust the next pointers

of adjacent nodes.

Traverse the list to find a node by its lineID or lineName and display its

4. Search for Assembly Line

details.

5. Update Assembly Line Status

Locate the node by lineID and update its status field.

6. Display All Assembly Lines

Traverse the list and print the details of each node.

Sample Menu

Menu:

1. Create Linked List of Assembly Lines

2. Insert New Assembly Line

3. Delete Assembly Line

4. Search for Assembly Line

5. Update Assembly Line Status

6. Display All Assembly Lines

7. Exit

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct AssemblyLine {
int lineID:
char lineName[50];
int capacity;
char status [20];
struct AssemblyLine* next;
} AssemblyLine;
AssemblyLine* createNode(int lineID, char* lineName, int capacity, char* status) {
AssemblyLine* newNode = (AssemblyLine*)malloc(sizeof(AssemblyLine));
newNode->lineID = lineID;
strcpy(newNode->lineName, lineName);
newNode->capacity = capacity;
strcpy(newNode->status, status);
newNode->next = NULL;
return newNode;
}
void append(Assembly Line** head, int lineID, char* lineName, int capacity, char*
status) {
AssemblyLine* newNode = createNode(lineID, lineName, capacity, status);
if (*head == NULL) {
```

```
*head = newNode;
return;
}
AssemblyLine* temp = *head; while (temp->next != NULL) {
temp = temp->next;
}
temp->next = newNode;
}
void display(Assembly Line* head) {
Assembly Line* temp = head;
while (temp != NULL) {
printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n", temp->lineID, temp- >lineName, temp->capacity, temp->status);
temp = temp->next;
}
}
void deleteById (AssemblyLine** head, int lineID) {
AssemblyLine* temp = *head;
AssemblyLine* prev = NULL;
while (temp != NULL && temp->lineID != lineID) { prev = temp;
temp = temp->next;
} if (temp == NULL) return; if (prev == NULL) { *head = temp->next; }
} else { prev->next = temp->next;
free(temp);
```

```
}

AssemblyLine* searchById(AssemblyLine* head, int lineID) {

AssemblyLine* temp = head;

while (temp != NULL) { temp = temp->next; }

if (temp->lineID == lineID) return temp;

return NULL;

}

void updateStatus (AssemblyLine* head, int lineID, char* newStatus) { AssemblyLine* temp = searchById(head, lineID);

if (temp != NULL) {

strcpy(temp->status, newStatus);

}

}

int main() {

AssemblyLine* head = NULL;

int choice, lineID, capacity;

char lineName[50], status [20], newStatus[20];

do {

printf("\nMenu:\n");

printf("1. Create Assembly Line\n");

printf("2. Insert New Assembly Line\n");

printf("3. Delete Assembly Line\n");

printf("4. Search Assembly Line\n");

printf("5. Update Assembly Line Status\n");

printf("6. Display All Assembly Lines\n");
printf("7. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter Line ID: ");

scanf("%d", &lineID);

printf("Enter Line Name: ");

scanf("%s", lineName);

printf("Enter Capacity: ");

scanf("%d", &capacity);

printf("Enter Status: ");

scanf("%s", status);

append(&head, lineID, lineName, capacity, status);

break;

case 2:

printf("Enter Line ID: ");

scanf("%d", &lineID);

printf("Enter Line Name: ");

scanf("%s", lineName);

printf("Enter Capacity: ");

scanf("%d", &capacity);

printf("Enter Status: ");

scanf("%s", status);
```

```
append(&head, lineID, lineName, capacity, status);

break;

case 3:

printf("Enter Line ID to delete: ");

scanf("%d", &lineID);

deleteById(&head, lineID);

break;

case 4:

printf("Enter Line ID to search: ");

scanf("%d", &lineID);

AssemblyLine* result = searchById(head, lineID);

if (result != NULL) {

printf("Line ID: %d, Name: %s, Capacity: %d, Status: %s\n", result->lineID,

result->lineName, result->capacity, result->status);

} else {

printf("Assembly line not found.\n");

}

break;

case 5:

printf("Enter Line ID to update: ");

scanf("%d", &lineID);

printf("Enter new status: ");

scanf("%s", newStatus);

updateStatus(head, lineID, newStatus);
```

```
break;
case 6:
display(head);
break;
case 7:
printf("Exiting program.\n");
break;
default:
printf("Invalid choice. Try again.\n");
}
} while (choice != 7);
return 0;
}
```

3. Implementation of stack using array

```
#include <stdio.h> #include <stdlib.h>
#define SUCCESS 0 #define FAILURE -1
typedef struct stack {
int capacity; int top; int *stack; } Stack_t;
int create_stack(Stack_t , int); int Push(Stack_t *, int); int Pop(Stack_t *); int Peek(Stack_t);
void Peep(Stack_t); int Peekindex(Stack_t stk, int index);
int main() { int choice, element, peek, size, index; Stack_t stk;
printf("Enter the size of the stack: "); scanf("%d", &size);
if (create_stack(&stk, size) == FAILURE) { printf("Error: Stack creation failed.\n"); return
FAILURE; }
```

```
while (1) { printf("\n1. Push\n2. Pop\n3. Display Stack\n4. Peek(Element at Top)\n5.
Peek(Element by index)\n6. Exit\nEnter your choice: ");
scanf("%d", &choice);

switch(choice) {

case 1:

printf("Enter the element to be pushed in stack: ");
scanf("%d", &element);
if (Push(&stk, element) == FAILURE) {
printf("INFO: Stack Full\n");

} break;

case 2:

if (Pop(&stk) == FAILURE) {
printf("INFO: Stack is empty\n");

} else {

printf("INFO: Pop operation is successful\n");

}

break;

case 3:

Peep(stk);

break;

case 4:

if ((peek = Peek(&stk)) == FAILURE) {

printf("INFO: Stack is empty\n");

} else {

printf("INFO: Peek element is %d\n", peek);

}

break;
case 5:
printf("Enter the index: ");
scanf("%d", &index);
if (Peekindex(stk, index) != FAILURE)
printf("The element at index %d is: %d\n", index, Peekindex(stk, index));
break;
case 6:
return SUCCESS;
default:
printf("Invalid Choice.\n");
break;
}
}
return 0;
}
int create_stack(Stack_t *stk, int size) {
stk->stack = (int*)malloc(size * sizeof(int));
if (stk->stack == NULL) {
return FAILURE;
}
stk->top = -1;
stk->capacity = size;
return SUCCESS;
}
```

```
int Push(Stack_t *stk, int element) {

if (stk->top == stk->capacity-1)

return FAILURE;

stk->top++;

stk->stack[stk->top] = element;

return SUCCESS;

}
int Pop(Stack_t *stk) {
if (stk->top == -1)
  return FAILURE;
stk->top--;
return SUCCESS;
}

int Peek(Stack_t *stk) {
if (stk->top == -1)
return FAILURE;
return stk->stack[stk->top];
}

int Peekindex(Stack_t stk, int index) {
if (stk.top == -1 || index < 0 || index > stk.top) {
printf("Invalid position!!\n");
return FAILURE;
}
return stk.stack[index];
}

void Peep(Stack_t stk) {
if (stk.top == -1) {
printf("Stack is empty!!\n");
return;
}
printf("Top -> ");
for (int i = 0; i <= stk.top; i++) {
printf("%d", stk.stack[i]);
}
printf("\n");
}
```