Week 2

Section 1 : Coding

1)

```
#include <stdio.h>
#include <stdlib.h>
Struct Node {
    Char item;
    Struct Node* next;
    Struct Node* prev;
};
// You are using Gcc
Void insertAtEnd (Node** head, char ch) {
    Node* newNode = (Node*) malloc (sizeof(Node));
    newNode -> item = ch;
    newNode -> next = NULL;
    newNode -> prev = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    While (temp -> next != NULL)
        temp = temp -> next;
    temp -> next = newNode;
    newNode -> prev = temp;
}
```

```
// Display forward
Void displayForward (Node *head) {
    While (head != NULL) {
        printf ("%c", head->item);
        head = head->next;
    }
    printf("\n");
}

// Display backward
Void displayBackward (Node *tail) {
    While (tail != NULL) {
        printf("%c", tail->item);
        tail = tail->prev;
    }
    printf("\n");
}

// Free the entire playlist
Void freeplaylist (Node *head) {
    Node *temp;
    While (head != NULL) {
        temp = head;
        head = head->next;
        free (temp);
    }
}
```

```
int main() {
    Struct Node* playlist =NULL;
    char item;
    While (1) {
        Scanf ("%c", &item);
        if (item == '-') {
            break;
        }
        insertAtEnd (&Playlist, item);
    }
    Struct Node* tail = playlist;
    While (tail->next != NULL) {
        tail = tail->next;
    }
    Printf ("Forward playlist:");
    displayForward (playlist);
    Printf ("Backward Playlist :");
    displayBackward (tail);
    freePlaylist (Playlist);
    Return 0;
}
```

Section 1 : coding

1)

```
// You are using Gcc
# include <stdio.h>
# include <Stdlib.h>
typedef Struct Node {
   int data;
   Struct Node* prev;
   Struct Node* next;
} Node;
typedef Struct DoublyLinkedList {
    Node* head;
    Node* tail;
 } DoublyLinkedList;
Void append (DoublyLinked List *list, int data) {
   Node* newNode = (Node*) malloc (sizeof (Node));
   newNode → data = data;
   newNode → prev = NULL;
   newNode → next = NULL;
   if (list → tail == NULL) { // if list is empty
      list → head = list → tail = newNode;
   } else {
     list → tail → next = newNode;
     newNode → prev = list → tail;
     list → tail = newNode;
   }
}
```

```
int findMax(DoublyLinkedList* list) {
    if(list->head == NULL) {
        Printf("Empty list\n");
        Return -1;
    }
    Node* current = list->head;
    int maxID = current->data;
    While(current != NULL) {
        if(current->data > maxID) {
            maxID = current->data;
        }
        current = current->next;
    }
    Return maxID;
}
int main() {
    DoublyLinkedList list = {NULL, NULL};
    int n, id;
    Scanf("%d", &n);
    if(n<1 || n>20) {
        Printf("Empty List!\n");
        Return 0;
    }
    for(int i=0; i<n; i++) {
        Scanf("%d", &id);
        if(id<1 || id>10000000) {
            Printf("Invalid ID!\n");
            Return 0;
        }
```

```
        append(&list, id);
    }
    int maxID = findMax(&list);
    if (maxID != -1) {
        printf("%d\n", maxID);
    }
    return 0;
}
```

Section 1 : coding

1)

```
#include<iostream>
using namespace std;

struct node {
   int info;
   struct node* prev, *next;
};
struct node* start = NULL;
Void traverse() {
    Struct node* temp = start;
    while (temp != NULL) {
        Printf ("%d", temp-> info);
        temp= temp->next;
    }
   Printf ("\n");
}
Void insertAtFront (int data) {
    Struct node* newNode (= (struct node*) malloc(sizeof
                                                (struct node));
    newNode -> info=data;
    newNode ->PREV= NULL;
    newNode -> next= start;
    if (start != NULL) {
        Start ->prev = newNode;
    }
```

```
        start = newNode;
        printf("Node Inserted\n");
}
int main() {
    int n, data;
    cin>>n;
    for(int i=0; i<n; i++){
        cin>>data;
        insertAtFront(data);
        traverse();
    }
    return 0;
}
```

Section 1 : coding

1)

```
// you are using Gcc
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    Struct Node* prev;
    Struct Node* next;
} Node;
typedef struct DoublyLinkedList {
    Node* head;
    Node* tail;
} DoublyLinkedList;
Void append(DoublyLinkedList* list, int data) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    if (list->tail == NULL) { // if the list is empty
        list->head = list->tail = newNode;
    } else {
        list->tail->next = newNode;
        newNode->prev = list->tail;
        list->tail = newNode;
    }
}
```

```
Void display(DoublyLinkedList* list) {
    Node* current = list->head;
    While (current != NULL) {
        Printf("%d", current->data);
        Current = Current->next;
    }
    Printf ("\n");
}
 int main() {
     DoublyLinkedList = {NULL, NULL};
     int n, id;
     Scanf("%d", &n);
     if (n<1 || n>10) {
         Printf ("Invalid input size!\n");
         Return 0;
    }
     for(int i=0; i<n; i++) {
         Scanf("%d", &id);
         if (id<1 || id>1000000) {
             Printf ("Invalid ID!\n");
             Return 0;
         }
        append(&list, id);
    }
     display (&list);
     Return 0;
}
```

Section 1: coding

1)

```
// You are using Gcc
# include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* Prev;
    Struct Node *next;
} Node;
// Function to insert at the end
Void insertAtEnd(Node** head, int value) {
    Node* newNode = (Node*) malloc (sizeof(Node));
    newNode -> data = value;
    newNode -> next = NULL;
    newNode -> Prev = NULL;
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    Node* temp = *head;
    While(temp -> next != NULL)
        temp = temp -> next;
    temp -> next = newNode;
    newNode -> prev = temp;
}
```

```
// Function to display the list
Void displayList(Node*head) {
    int index=1;
    while(head!=NULL) {
        printf("node %d: %d\n", index++, head->data);
        head=head->next;
    }
}
// Function to display the list
Void displayList(Node*head) {
    int index=1;
    While(head!=NULL) {
        printf("node %d : %d\n", index++, head->data);
        head=head->next;
    }
}
// Function to delete node at a given 1-based position
int deleteAtposition(Node**head, int pos) {
    if (*head==NULL || Pos<=0)
      return 0;
    Node*temp=*head;
    int count=1;
    While (temp!=NULL && count<Pos) {
        temp=temp->next;
        count++;
    }
    if (temp==NULL)
        Return 0; // Invalid position
```

```
    if (temp->prev!=NULL)
        temp->prev->next=temp->next;
    else
        *head=temp->next; //Deleting head
    if (temp->next!=NULL)
        temp->next->prev=temp->prev;
    free(temp);
    return 1: // successfully deleted
}
//Free memory
Void freeList(Node*head){
    Node*temp;
    While(head!=NULL){
        temp=head;
        head=head->next;
        free(temp);
    }
}
int main(){
    Node*head=NULL;
    int n, value, pos;
    Scanf("%d", &n);
    for(int i=0; i<n; i++){
        Scanf("%d", &value);
        insertAtEnd(&head, value);
    }
    Scanf("%d", &pos);
```

```
    printf("Data entered in the list:\n");
    displayList(head);
    if(!deleteAtPosition(&head, pos)) {
        printf("Invalid position. Try again.\n");
    } else {
        printf("After deletion the new list:\n");
        displayList(head);
    }
    freeList(head);
    return 0;
}
```