Name : Deshmukh Mehmood Rehan

MIS No. : 612303050

SY Computer Science – Div 1

Question: Define an ADT ASCII having a Doubly linked list of nodes of digits.

Write the following functions with suitable prototypes for ADT ASCII

init_ASCII() // to initialize the list

ASCII_of( )

/* This function should take a character as an argument and form a linked list of digits in its ASCII value. For example if 'A' is passed as a parameter, it should generate a list {6,5} */

traverse() // to display all the elements of the list.

Destroy() // to destroy the list.

You are free to include more functions.

The skeleton of function main() is given below, use the same by replacing commented statements by actual function calls:

```
int main() {

ASCII L1;

//call init_ASCII()

// call ASCII_of( )

// call traverse()

//call Destroy()

//call traverse()

return 0;

}
```

Code:

header.h : This File includes the declarations of structures and function prototypes

```
// this structure will be the node of our Doubly Linked  List;
// For it to be a "Doubly" LinkedList we need the node to have two pointers
// A previous pointer "prev" which will point to the previous element
// and a Next pointer "next" which will point to the next element


//in case of first Node the prev will point to NULL and
// in case of last element the next will point to NULL

typedef struct Node{
    int data; // the data (in our case it is an Integer as the ASCII values
will be stored as Integers)
    struct Node *next; // the next pointer
    struct Node *prev; // the prev pointer
} Node;

// This will be our Data Structure or more specifically the ADT
// for that we will need two pointers, first one to the beginning of the list
and
// other at the end of the list
// we can typedef them into a struct which is our Ascii ADT.

typedef struct ascii{
    Node *head; //pointer to the first Node
    Node *tail; // pointer to the last Node
} Ascii;

//These are the functions mentioned in the assignment to implement :

void initAscii(Ascii *l); //  // this function initializes the list
void asciiOf(Ascii *l, char ch);/* This function takes a character as an
argument and form a linked list of digits in its ASCII value. For example if
'A' is passed as a parameter, it should generate a list {6,5} */
void traverse(Ascii l); // this function displays all the elements of the
list.
void destroy(Ascii *l); //  this function destroys the list.




int isEmpty(Ascii l);
int length(Ascii l);
```

```
void append(Ascii *l, int data);
void insertAtStart(Ascii *l, int data);
void insertAtIndex(Ascii *l, int data, int index);
int removeStart(Ascii *l);
int removeAtIndex(Ascii *l, int index);
int removeAtEnd(Ascii *l);
void reverseList(Ascii *l);
```

logic.c : contains the definition of all the functions declared in the header file along with some helper functions

```
#include <stdio.h>
#include "header.h" //including the header file
#include <stdlib.h> // the C standard Library for functions like malloc
#include <limits.h> // includes constants like INT_MIN / INT_MAX

// to initialize our ADT we need a notion of an empty list which is possible
when both
// the head pointer and the tail pointer point to NULL
// so we will initialize our ADT by making the head and tail pointers point to
NULL
void initAscii(Ascii *l) {
    l->head = l->tail = NULL; //both head and tail point to NULL
    return;
}

// to generate a list of digits in the ASCII value of a character we will
follow the following logic
// typecast to int to get the ascii value;
// repeat untill ascii value > 0 :
        // get the last digit by taking modulo with 10
        // insert that digit in the beginning
        // divide the ascii value by 10

//to insert a Node at beginning we will use a helper function insertAtStart

void asciiOf(Ascii *l, char ch){
    int asciiValue = (int)ch; //get the ASCII value

    // get the last digit and insert at beginning
    while(asciiValue){
        insertAtStart(l, asciiValue % 10);
        asciiValue /= 10;
    }
```

```
    return;
}

//to traverse and display the list we need a temporary pointer which will
point to the
// first node . while the temp is not pointing to null we will print the
node's data
// and increment temp to point to the next Node

//if the list is empty we simply return
void traverse(Ascii l) {
    if (isEmpty(l)) //if list is empty .. return
        return;

    printf("Displaying the LinkedList: ");

    //traverse the entire list
    for (Node *temp = l.head; temp; temp = temp->next)
    {
        printf("%d <-> ", temp->data);
    }

    //get rid of the extra "<-> " at the end and print a newline
    printf("\b\b\b\b     \n");

    return;
}

//to destroy the list, we repeatedly remove the first Node from the list
untill the list is empty

//we will use a helper function to remove the Node at the beginning
void destroy(Ascii *l){
    if(isEmpty(*l)) return; //if list is empty.. return

    // repeatedly remove the last element of the array untill the list is
empty
    while(!isEmpty(*l)){
        removeStart(l);
    }

    return;
}

// helper functions

//checks if the list is empty
```

```c
// if the head is pointing to NULL, it means that there is not a single Node
in the list
// we can use tail too in this function
int isEmpty(Ascii l) {
    return (!l.head); //returns 1 if head is pointing to NULL else 0
}

// this is the helper function used to add a Node to the beginning of the list
// newNode's next will point to l->head and the new head will be the newNode
//  we also handle the case of empty list by making the head and tail point to
the newNode
void insertAtStart(Ascii *l, int data){
    Node *newNode = (Node *)malloc(sizeof(Node)); //allocate memory for new
Node
    if(!newNode) return; // if not allocated ..return

    newNode->data = data;
    newNode->next = newNode->prev = NULL;

    if(isEmpty(*l)){
        l->head = l->tail = newNode; //handling case of empty list
        return;
    }

    // newNode's next will point to l->head and the new head will be new node
    newNode->next = l->head;
    l->head->prev = newNode;
    l->head = newNode;

    return;
}

//this is the helper function we used in the destroy function
// this function frees the head and the head's next becomes the new head
// we also handle the case of empty list by makin g l->tail point to NULL
int removeStart(Ascii *l){
    if(isEmpty(*l)) return INT_MIN;
    Node *removedNode;
    int removedElement;

    removedNode = l->head; //Node to be removed
    removedElement = removedNode->data; //element which we will return

    l->head = removedNode->next;
    if(isEmpty(*l)){
        l->tail = NULL; //handling empty list case
    }else {
        l->head->prev = NULL;
```

```
    }

    free(removedNode); //freeing the Node

    return removedElement;
}


//Following are some more helper function which I implemented.
// they can be used to manipulate Doubly Linked Lists
void append(Ascii *l, int data) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    if (!newNode)
        return;

    newNode->data = data;
    newNode->next = newNode->prev = NULL;

    if (isEmpty(*l))
    {
        l->head = l->tail = newNode;
        return;
    }

    l->tail->next = newNode;
    newNode->prev = l->tail;
    l->tail = newNode;

    return;
}

int length(Ascii l) {
    int len = 0;
    for (Node *temp = l.head; temp; temp = temp->next)
    {
        len++;
    }

    return len;
}



void insertAtIndex(Ascii *l, int data, int index){
    if(index < 0 || index > length(*l)) return;

    if(index == 0){
        insertAtStart(l, data);
        return;
```

```
    }

    Node *newNode = (Node *)malloc(sizeof(Node));
    if(!newNode) return;

    newNode->data = data;
    newNode->next = newNode->prev = NULL;

    Node *temp = l->head;

    for(int i = 0; i < index - 1 && temp; i++){
        temp = temp->next;
    }

    newNode->next = temp->next;
    if (temp->next) temp->next->prev = newNode;
    temp->next = newNode;
    newNode->prev = temp;

    if (newNode->next == NULL) {
        l->tail = newNode;
    }

    return;

}




int removeAtIndex(Ascii *l, int index){
    if(isEmpty(*l) || index < 0 || index > length(*l)) return INT_MIN;
    if(index == 0){
        return removeStart(l);
    }

    Node *removedNode, *temp = l->head;
    int removedElement;

    for(int i = 0; i < index - 1; i++){
        temp = temp->next;
    }

    removedNode = temp->next;
    temp->next = removedNode->next;
    if(removedNode->next){
        removedNode->next->prev = temp;
    }else{
        l->tail = temp;
```

```
    }

    removedElement = removedNode->data;
    free(removedNode);

    return removedElement;
}

int removeAtEnd(Ascii *l){
    if(isEmpty(*l)) return INT_MIN;

    Node *removedNode;
    int removedElement;

    removedNode = l->tail;
    removedElement = removedNode->data;
    l->tail = l->tail->prev;
    if(!l->tail){
        l->head = NULL;
    }else{
        l->tail->next = NULL;
    }


    free(removedNode);
    return removedElement;
}

void reverseAsciiList(Ascii *l){
    Node *curr, *next, *prev, *temp;
    prev = NULL;
    temp = curr = l->head;

    while (curr){
        next = curr->next;
        curr->next = prev;
        curr->prev = next;
        prev = curr;
        curr = next;
    }

    l->head = prev;
    l->tail = temp;
    return;
}
```

main.c : This contains the code to test the implementation

```c
#include "logic.c"

int main(){
    Ascii l1;
    initAscii(&l1);
    asciiOf(&l1, 'A');
    traverse(l1);
    destroy(&l1);
    traverse(l1); // should not print anything
    Ascii l2;
    initAscii(&l2);
    asciiOf(&l2, 'a');
    traverse(l2);
    destroy(&l2);
    traverse(l2); //should not print anything

    return 0;
}
```

Output:

```
Assignment 2>gcc .\main.c -Wall -o main
Assignment 2>.\main.exe
Displaying the LinkedList: 6 <-> 5
Displaying the LinkedList: 9 <-> 7
Assignment 2>
```