ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2017-2018

ΧΕΙΜΕΡΙΝΟ ΕΞΑΜΗΝΟ

ΜΑΘΗΜΑ: ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΕΡΓΑΣΤΗΡΙΟ 4 – ΣΗΜΕΙΩΣΕΙΣ

Εγγραφές (Structs)

Οι εγγραφές (structs) είναι μια συλλογή μεταβλητών κάτω από ένα κοινό όνομα. Ένα παράδειγμα δήλωσης μιας εγγραφής έχει ως εξής:

```
struct student
{
	int r_no;
	char name[20];
	char course[20];
	float fees;
};
```

Η δημιουργία ενός νέου τύπου δεδομένων στη C μπορεί να υλοποιηθεί με χρήση της λέξης κλειδί typedef.

```
typedef struct student
{
	int r_no;
	char name[20];
	char course[20];
	float fees;
};
```

Οπότε στη συνέχεια μπορεί να ακολουθήσει μια δήλωση ως εξής:

```
student stud1;
```

Η εκχώρηση τιμών σε κάθε πεδίο μιας εγγραφής μπορεί να γίνει με κάποιον από τους ακόλουθους τρόπους:

```
struct student stud1 = {01, "Rahul", "BCA", 45000};
```

ή

```
stud1.r_no = 01;
stud1.name = "Rahul";
stud1.course = "BCA";
stud1.fees = 45000;
```

Code 1

(ανάγνωση και εμφάνιση στοιχείων ενός συνόλου φοιτητών που αποθηκεύονται σε ένα πίνακα)

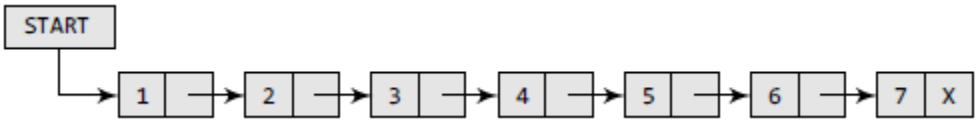
```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
```

```
{
        struct student
        {
                int roll_no;
                char name[80];
                int fees;
                char DOB[80];
        };
        struct student stud[50];

        int n, i, num, new_rolno;
        int new_fees;
        char new_DOB[80], new_name[80];
        printf("\n Enter the number of students : ");
        scanf("%d", &n);
        for(i=0;i<n;i++)
        {
                printf("\n Enter the roll number : ");
                scanf("%d", &stud[i].roll_no);
                printf("\n Enter the name : ");
                gets(stud[i].name);
                printf("\n Enter the fees : ");
                scanf("%d",&stud[i].fees);
                printf("\n Enter the DOB : ");
                gets(stud[i].DOB);
        }
        for(i=0;i<n;i++)
        {
                printf("\n ********DETAILS OF STUDENT %d*******", i+1);
                printf("\n ROLL No. = %d", stud[i].roll_no);
                printf("\n NAME = %s", stud[i].name);
                printf("\n FEES = %d", stud[i].fees);
                printf("\n DOB = %s", stud[i].DOB);
        }
        printf("\n Enter the student number whose record has to be edited : ");
        scanf("%d", &num);
        num= num–1;
        printf("\n Enter the new roll number : ");
        scanf("%d", &new_rolno);
        printf("\n Enter the new name : ");
        gets(new_name);
        printf("\n Enter the new fees : ");
        scanf("%d", &new_fees);
```

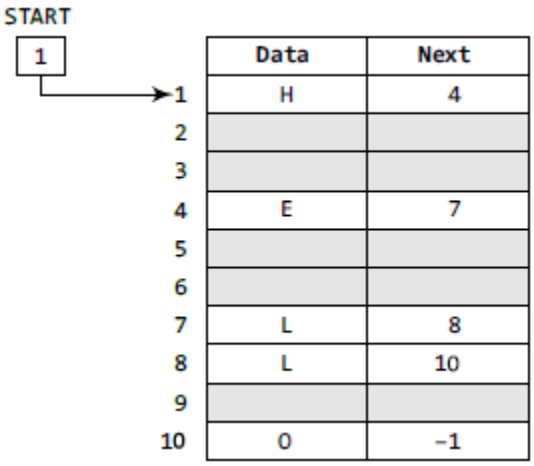
```
        printf("\n Enter the new DOB : ");
        gets(new_DOB);
        stud[num].roll_no = new_rolno;
        strcpy(stud[num].name, new_name);
        stud[num].fees = new_fees;
        strcpy (stud[num].DOB, new_DOB);
        for(i=0;i<n;i++)
        {
                printf("\n ********DETAILS OF STUDENT %d*******", i+1);
                printf("\n ROLL No. = %d", stud[i].roll_no);
                printf("\n NAME = %s", stud[i].name);
                printf("\n FEES = %d", stud[i].fees);
                printf("\n DOB = %s", stud[i].DOB);
        }
        getch();
        return 0;
}
```

Συνδεδεμένες Λίστες

Με απλά λόγια μια **συνδεδεμένη λίστα (linked list)** είναι μια γραμμική συλλογή στοιχείων τα οποία καλούνται κόμβοι. Οι λίστες χρησιμοποιούνται συνήθως για την υλοποίηση άλλων δομών δεδομένων όπως για παράδειγμα οι στοίβες ή οι ουρές. Σχηματικά μια λίστα έχει ως ακολούθως:



Η ακόλουθη εικόνα μας παρουσιάζει τον τρόπο με τον οποίο μια λίστα αποθηκεύεται στη μνήμη του συστήματος:



	Data	Next
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	0	-1

Κάθε κόμβος, στο παραπάνω παράδειγμα, περιλαμβάνει δύο τμήματα: ένα ακέραιο σαν τα δεδομένα του κόμβου και ένα δείκτη προς τον επόμενο κόμβο. Το τμήμα δεδομένων του κάθε κόμβου μπορεί να περιέχει οτιδήποτε π.χ., ακεραίους, πίνακες ή ακόμα και εγγραφές. Για τον τελευταίο κόμβο, ο δείκτης προς τον επόμενο κόμβο είναι ίσος με NULL. Τέλος, μια λίστα περιλαμβάνει ένα δείκτη START που δείχνει στον πρώτο κόμβο της λίστας.

Μπορούμε να διασχίσουμε τη λίστα ξεκινώντας από το δείκτη START και προσπελάζοντας τον κάθε κόμβο μέσω του δείκτη του προηγούμενου κόμβου. Μια λίστα μπορεί να οριστεί με την ακόλουθη δομή:

```
struct node
{
        int data;
        struct node *next;
};
```

Οι λίστες, όπως και οι πίνακες, είναι γραμμικές συλλογές δεδομένων αλλά όμως οι λίστες δεν αποθηκεύονται σε συνεχόμενες θέσεις στη μνήμη. Μια άλλη διαφορά είναι ότι οι λίστες δεν επιτρέπουν την τυχαία προσπέλαση διαφόρων κόμβων. Οι κόμβοι μπορούν να προσπελαστούν μόνο σειριακά. Αντίθετα με τους πίνακες όμως, οι εισαγωγές και οι διαγραφές μπορούν να γίνουν σε οποιοδήποτε σημείο της δομής σε σταθερό χρόνο. Τέλος, δεν υπάρχει περιορισμός στο πλήθος των στοιχείων που μπορούν να προστεθούν σε μια λίστα.

Ενέργειες πάνω σε μια Απλά Συνδεδεμένη Λίστα

Οι ενέργειες που μπορούν να γίνουν πάνω σε μια λίστα έχουν ως εξής:

- **Διάσχιση**. Αποκτούμε πρόσβαση σε κάθε κόμβο ξεκινώντας από την κορυφή (START).

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:           Apply Process to PTR->DATA
Step 4:           SET PTR = PTR->NEXT
       [END OF LOOP]
Step 5: EXIT
```

- **Αναζήτηση**. Ψάχνουμε να βρούμε ένα συγκεκριμένο στοιχείο μέσα στη λίστα.

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Step 3 while PTR != NULL
Step 3:     IF VAL = PTR->DATA
                  SET POS = PTR
                  Go To Step 5
            ELSE
                  SET PTR = PTR->NEXT
            [END OF IF]
       [END OF LOOP]
Step 4: SET POS = NULL
Step 5: EXIT
```

- **Εισαγωγή**. Η εισαγωγή ενός στοιχείου σε μια λίστα περιλαμβάνει τη δημιουργία ενός νέου κόμβου καθώς και την εισαγωγή του στη σωστή θέση μέσα στη λίστα. Ισχύουν οι ακόλουθες περιπτώσεις:
  - Η εισαγωγή γίνεται στην **αρχή** της λίστας

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET START = NEW_NODE
Step 7: EXIT
```

- Η εισαγωγή γίνεται στο **τέλος** της λίστας

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

- Η εισαγωγή γίνεται **μετά** από ένα δοσμένο κόμβο

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PREPTR -> DATA
          != NUM
Step 8:     SET PREPTR = PTR
Step 9:     SET PTR = PTR -> NEXT
          [END OF LOOP]
Step 10: PREPTR -> NEXT = NEW_NODE
Step 11: SET NEW_NODE -> NEXT = PTR
Step 12: EXIT
```

- Η εισαγωγή γίνεται **πριν** από ένα δοσμένο κόμβο

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL->NEXT
Step 4: SET NEW_NODE->DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR->DATA != NUM
Step 8:      SET PREPTR = PTR
Step 9:      SET PTR = PTR->NEXT
         [END OF LOOP]
Step 10: PREPTR->NEXT = NEW_NODE
Step 11: SET NEW_NODE->NEXT = PTR
Step 12: EXIT
```

- **Διαγραφή.** Διαγράφουμε ένα συγκεκριμένο κόμβο από τη λίστα. Ισχύουν οι ακόλουθες περιπτώσεις:
  - Διαγράφεται ο **πρώτος** κόμβος της λίστας

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 5
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START->NEXT
Step 4: FREE PTR
Step 5: EXIT
```

  - Διαγράφεται ο **τελευταίος** κόμβος

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR->NEXT != NULL
Step 4:      SET PREPTR = PTR
Step 5:      SET PTR = PTR->NEXT
        [END OF LOOP]
Step 6: SET PREPTR->NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

  - Διαγράφεται ο **επόμενος** κόμβος ενός δοσμένου κόμβου

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR -> DATA != NUM
Step 5:     SET PREPTR = PTR
Step 6:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR -> NEXT = PTR -> NEXT
Step 9: FREE TEMP
Step 10: EXIT
```

Code 2

(Υλοποιεί με τη χρήση ξεχωριστών συναρτήσεων όλες τις λειτουργίες σε μια λίστα)

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>
struct node
{
        int data;
        struct node *next;
};
struct node *start = NULL;
struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node *);
struct node *insert_after(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
struct node *sort_list(struct node *);

int main(int argc, char *argv[]) {
        int option;
        do
        {
                printf("\n\n *****MAIN MENU *****");
                printf("\n 1: Create a list");
```

```
printf("\n 2: Display the list");
printf("\n 3: Add a node at the beginning");
printf("\n 4: Add a node at the end");
printf("\n 5: Add a node before a given node");
printf("\n 6: Add a node after a given node");
printf("\n 7: Delete a node from the beginning");
printf("\n 8: Delete a node from the end");
printf("\n 9: Delete a given node");
printf("\n 10: Delete a node after a given node");
printf("\n 11: Delete the entire list");
printf("\n 12: Sort the list");
printf("\n 13: EXIT");
printf("\n\n Enter your option : ");
scanf("%d", &option);
switch(option)
{
        case 1: start = create_ll(start);
                printf("\n LINKED LIST CREATED");
                break;
        case 2: start = display(start);
                break;
        case 3: start = insert_beg(start);
                break;
        case 4: start = insert_end(start);
                break;
        case 5: start = insert_before(start);
                break;
        case 6: start = insert_after(start);
                break;
        case 7: start = delete_beg(start);
                break;
        case 8: start = delete_end(start);
                break;
        case 9: start = delete_node(start);
                break;
        case 10: start = delete_after(start);
                break;
        case 11: start = delete_list(start);
                printf("\n LINKED LIST DELETED");
                break;
        case 12: start = sort_list(start);
                break;
}
```

```
        }while(option !=13);
        getch();
        return 0;
}

struct node *create_ll(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter -1 to end");
        printf("\n Enter the data : ");
        scanf("%d", &num);
        while(num!=-1)
        {
                new_node = (struct node*)malloc(sizeof(struct node));
                new_node -> data=num;
                if(start==NULL)
                {
                        new_node -> next = NULL;
                        start = new_node;
                }
                else
                {
                        ptr=start;
                        while(ptr->next!=NULL)
                                ptr=ptr->next;
                        ptr->next = new_node;
                        new_node->next=NULL;
                }
                printf("\n Enter the data : ");
                scanf("%d", &num);
        }
        return start;
}

struct node *display(struct node *start)
{
        struct node *ptr;
        ptr = start;
        while(ptr != NULL)
        {
                printf("\t %d", ptr -> data);
                ptr = ptr -> next;
```

```
        }
        return start;
}

struct node *insert_beg(struct node *start)
{
        struct node *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        new_node -> next = start;
        start = new_node;
        return start;
}

struct node *insert_end(struct node *start)
{
        struct node *ptr, *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        new_node -> next = NULL;
        ptr = start;
        while(ptr -> next != NULL)
                ptr = ptr -> next;
        ptr -> next = new_node;
        return start;
}

struct node *insert_before(struct node *start)
{
        struct node *new_node, *ptr, *preptr;
        int num, val;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        printf("\n Enter the value before which the data has to be inserted : ");
        scanf("%d", &val);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
```

```
        ptr = start;
        while(ptr -> data != val)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next = new_node;
        new_node -> next = ptr;
        return start;
}

struct node *insert_after(struct node *start)
{
        struct node *new_node, *ptr, *preptr;
        int num, val;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        printf("\n Enter the value after which the data has to be inserted : ");
        scanf("%d", &val);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr = start;
        preptr = ptr;
        while(preptr -> data != val)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next=new_node;
        new_node -> next = ptr;
        return start;
}

struct node *delete_beg(struct node *start)
{
        struct node *ptr;
        ptr = start;
        start = start -> next;
        free(ptr);
        return start;
}

struct node *delete_end(struct node *start)
```

```
{
        struct node *ptr, *preptr;
        ptr = start;
        while(ptr -> next != NULL)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next = NULL;
        free(ptr);
        return start;
}

struct node *delete_node(struct node *start)
{
        struct node *ptr, *preptr;
        int val;
        printf("\n Enter the value of the node which has to be deleted : ");
        scanf("%d", &val);
        ptr = start;
        if(ptr -> data == val)
        {
                start = delete_beg(start);
                return start;
        }
        else
        {
                while(ptr -> data != val)
                {
                        preptr = ptr;
                        ptr = ptr -> next;
                }
                preptr -> next = ptr -> next;
                free(ptr);
                return start;
        }
}

struct node *delete_after(struct node *start)
{
        struct node *ptr, *preptr;
        int val;
        printf("\n Enter the value after which the node has to deleted : ");
```

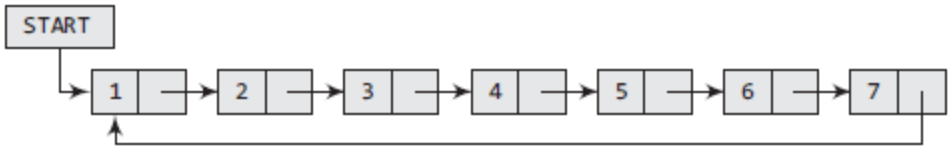
```
        scanf("%d", &val);
        ptr = start;
        preptr = ptr;
        while(preptr -> data != val)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next=ptr -> next;
        free(ptr);
        return start;
}

struct node *delete_list(struct node *start)
{
        struct node *ptr; // Lines 252-254 were modified from original code to fix
        unresposiveness in output window
        if(start!=NULL){
                ptr=start;
                while(ptr != NULL)
                {
                        printf("\n %d is to be deleted next", ptr -> data);
                        start = delete_beg(ptr);
                        ptr = start;
                }
        }
        return start;
}
struct node *sort_list(struct node *start)
{
        struct node *ptr1, *ptr2;
        int temp;
        ptr1 = start;
        while(ptr1 -> next != NULL)
        {
                ptr2 = ptr1 -> next;
                while(ptr2 != NULL)
                {
                        if(ptr1 -> data > ptr2 -> data)
                        {
                                temp = ptr1 -> data;
                                ptr1 -> data = ptr2 -> data;
                                ptr2 -> data = temp;
```

```
                    }
                    ptr2 = ptr2 -> next;
                }
                ptr1 = ptr1 -> next;
            }
            return start; // Had to be added
}
```

Κυκλικές Λίστες

Σε μια κυκλική λίστα, ο τελευταίος κόμβος περιέχει ένα δείκτη προς τον πρώτο κόμβο της λίστας αντί να δείχνει προς μια τιμή NIL/NULL. Μπορούμε να έχουμε κυκλικές απλές συνδεδεμένες λίστες ή κυκλικές διπλά συνδεδεμένες λίστες. Κατά τη διάσχιση μιας κυκλικής λίστας μπορούμε να ξεκινήσουμε από οποιοδήποτε κόμβο μέχρι που να φτάσουμε στον κόμβο από τον οποίο ξεκινήσαμε. Στις διπλά συνδεδεμένες λίστες η κατεύθυνση διάσχισης μπορεί να είναι οποιαδήποτε.

Τα επόμενα σχήματα μας απεικονίζουν ένα παράδειγμα μιας κυκλικής λίστας καθώς και τον τρόπο αποθήκευσης στη μνήμη.



START = 1

	DATA	NEXT
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	O	1

Η μόνη δυσκολία διαχείρισης μιας κυκλικής λίστας είναι η πολυπλοκότητα της διάσχισης αφού δεν υπάρχει δείκτης προς την τιμή NIL/NULL που να καθορίζει το τέλος της λίστας. Παρά το γεγονός αυτό, μπορούμε να διασχίσουμε τη λίστα μέχρι να συναντήσουμε τον κόμβο που είναι στην κορυφή, πράγμα το οποίο σηματοδοτεί το τέλος της λίστας. Ακολουθούν βασικοί αλγόριθμοι διαχείρισης μιας κυκλικής λίστας και ο αντίστοιχος κώδικας για κάθε ένα από αυτούς.

- **Εισαγωγή κόμβου στην αρχή της λίστας**

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:      PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = START
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET START = NEW_NODE
Step 11: EXIT
```

- **Εισαγωγή ενός κόμβου στο τέλος της λίστας**

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 10
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:      SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

- **Διαγραφή του πρώτου κόμβου της λίστας**

```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 8
          [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4:       SET PTR = PTR -> NEXT
         [END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: FREE START
Step 7: SET START = PTR -> NEXT
Step 8: EXIT
```

- **Διαγραφή του τελευταίου κόμβου της λίστας**

```
Step 1: IF START = NULL
              Write UNDERFLOW
              Go to Step 8
         [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != START
Step 4:         SET PREPTR = PTR
Step 5:         SET PTR = PTR -> NEXT
         [END OF LOOP]
Step 6: SET PREPTR -> NEXT = START
Step 7: FREE PTR
Step 8: EXIT
```

Code 3

(Υλοποιεί με τη χρήση ξεχωριστών συναρτήσεων όλες τις λειτουργίες σε μια κυκλική λίστα)

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node
{
        int data;
        struct node *next;
};
struct node *start = NULL;

struct node *create_cll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);

int main()
{
        int option;
        do
        {
                printf("\n\n *****MAIN MENU *****");
                printf("\n 1: Create a list");
                printf("\n 2: Display the list");
                printf("\n 3: Add a node at the beginning");
                printf("\n 4: Add a node at the end");
                printf("\n 5: Delete a node from the beginning");
                printf("\n 6: Delete a node from the end");
```

```
                printf("\n 7: Delete a node after a given node");
                printf("\n 8: Delete the entire list");
                printf("\n 9: EXIT");
                printf("\n\n Enter your option : ");
                scanf("%d", &option);
                switch(option)
                {
                        case 1: start = create_cll(start);
                                printf("\n CIRCULAR LINKED LIST CREATED");
                                break;
                        case 2: start = display(start);
                                break;
                        case 3: start = insert_beg(start);
                                break;
                        case 4: start = insert_end(start);
                                break;
                        case 5: start = delete_beg(start);
                                break;
                        case 6: start = delete_end(start);
                                break;
                        case 7: start = delete_after(start);
                                break;
                        case 8: start = delete_list(start);
                                printf("\n CIRCULAR LINKED LIST DELETED");
                                break;
                }
        }while(option !=9);
        getch();
        return 0;
}

struct node *create_cll(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter –1 to end");
        printf("\n Enter the data : ");
        scanf("%d", &num);
        while(num!=–1)
        {
                new_node = (struct node*)malloc(sizeof(struct node));
                new_node -> data = num;
                if(start == NULL)
```

```
                {
                        new_node -> next = new_node;
                        start = new_node;
                }
                else
                {
                        ptr = start;
                        while(ptr -> next != start)
                                ptr = ptr -> next;
                        ptr -> next = new_node;
                        new_node -> next = start;
                }
                printf("\n Enter the data : ");
                scanf("%d", &num);
        }
        return start;
}

struct node *display(struct node *start)
{
        struct node *ptr;
        ptr=start;
        while(ptr -> next != start)
        {
                printf("\t %d", ptr -> data);
                ptr = ptr -> next;
        }
        printf("\t %d", ptr -> data);
        return start;
}

struct node *insert_beg(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr = start;
        while(ptr -> next != start)
                ptr = ptr -> next;
        ptr -> next = new_node;
```

```
        new_node -> next = start;
        start = new_node;
        return start;
}

struct node *insert_end(struct node *start)
{
        struct node *ptr, *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr = start;
        while(ptr -> next != start)
                ptr = ptr -> next;
        ptr -> next = new_node;
        new_node -> next = start;
        return start;
}

struct node *delete_beg(struct node *start)
{
        struct node *ptr;
        ptr = start;
        while(ptr -> next != start)
                ptr = ptr -> next;
        ptr -> next = start -> next;
        free(start);
        start = ptr -> next;
        return start;
}

struct node *delete_end(struct node *start)
{
        struct node *ptr, *preptr;
        ptr = start;
        while(ptr -> next != start)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next = ptr -> next;
```

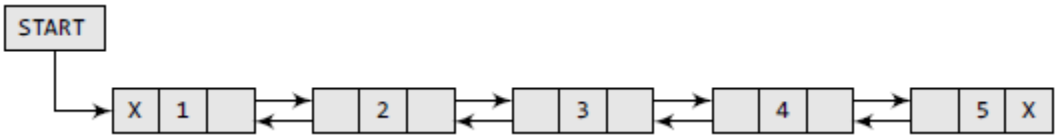
```
        free(ptr);
        return start;
}

struct node *delete_after(struct node *start)
{
        struct node *ptr, *preptr;
        int val;
        printf("\n Enter the value after which the node has to deleted : ");
        scanf("%d", &val);
        ptr = start;
        preptr = ptr;
        while(preptr -> data != val)
        {
                preptr = ptr;
                ptr = ptr -> next;
        }
        preptr -> next = ptr -> next;
        if(ptr == start)
                start = preptr -> next;
        free(ptr);
        return start;
}

struct node *delete_list(struct node *start)
{
        struct node *ptr;
        ptr = start;
        while(ptr -> next != start)
                start = delete_end(start);
        free(start);
        return start;
}
```

Διπλά Συνδεδεμένες Λίστες

Πρόκειται για ένα πιο πολύπλοκο τύπο λίστας αφού τώρα ο κάθε κόμβος έχει δύο δείκτες: ο ένας δείχνει προς τον επόμενο κόμβο ενώ ο άλλος δείχνει προς τον προηγούμενο. Με βάση αυτό το σκεπτικό, η δομή που υιοθετούμε για τον κάθε κόμβο έχει ως εξής:

```
struct node
{
        struct node *prev;
        int data;
        struct node *next;
};
```

ενώ σχηματικά μια διπλά συνδεδεμένη λίστα έχει ως ακολούθως:



Στη μνήμη, οι κόμβοι αποθηκεύονται όπως απεικονίζεται στο ακόλουθο παράδειγμα:

START = 1

	DATA	PREV	NEXT
1	H	-1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	-1

Παρά το γεγονός ότι αυτού του είδους οι λίστες απαιτούν περισσότερο χώρο για κάθε κόμβο, έχουν το πλεονέκτημα της διάσχισης και προς τις δύο κατευθύνσεις. Ακολουθούν οι αλγόριθμοι και ο κώδικας που υλοποιούντις βασικές λειτουργίες σε μια διπλά συνδεδεμένη λίστα.

- **Εισαγωγή κόμβου στην αρχή της λίστας**.

```
Step 1: IF AVAIL = NULL
                Write OVERFLOW
                Go to Step 9
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> PREV = NULL
Step 6: SET NEW_NODE -> NEXT = START
Step 7: SET START -> PREV = NEW_NODE
Step 8: SET START = NEW_NODE
Step 9: EXIT
```

- **Εισαγωγή κόμβου στο τέλος της λίστας**.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 11
       [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
       [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: EXIT
```

- **Εισαγωγή κόμβου μετά από ένα δοσμένο κόμβο**.

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR -> NEXT
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET PTR -> NEXT = NEW_NODE
Step 11: SET PTR -> NEXT -> PREV = NEW_NODE
Step 12: EXIT
```

- **Εισαγωγή κόμβου πριν από ένα δοσμένο κόμβο.**

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> DATA != NUM
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET NEW_NODE -> NEXT = PTR
Step 9: SET NEW_NODE -> PREV = PTR -> PREV
Step 10: SET PTR -> PREV = NEW_NODE
Step 11: SET PTR  -> PREV -> NEXT = NEW_NODE
Step 12: EXIT
```

- **Διαγραφή κόμβου στην αρχή της λίστας.**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 6
        [END OF IF]
Step 2: SET PTR = START
Step 3: SET START = START -> NEXT
Step 4: SET START -> PREV = NULL
Step 5: FREE PTR
Step 6: EXIT
```

- **Διαγραφή κόμβου στο τέλος της λίστας.**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 7
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != NULL
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = NULL
Step 6: FREE PTR
Step 7: EXIT
```

- **Διαγραφή κόμβου μετά από ένα δοσμένο κόμβο.**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:      SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> NEXT
Step 6: SET PTR -> NEXT = TEMP -> NEXT
Step 7: SET TEMP -> NEXT -> PREV = PTR
Step 8: FREE TEMP
Step 9: EXIT
```

- **Διαγραφή κόμβου πριν από ένα δοσμένο κόμβο.**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 9
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> DATA != NUM
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET TEMP = PTR -> PREV
Step 6: SET TEMP -> PREV -> NEXT = PTR
Step 7: SET PTR -> PREV = TEMP -> PREV
Step 8: FREE TEMP
Step 9: EXIT
```

Code 4

(Υλοποιεί με τη χρήση ξεχωριστών συναρτήσεων όλες τις λειτουργίες σε μια διπλά συνδεδεμένη λίστα)

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node
{
```

```
        struct node *next;
        int data;
        struct node *prev;
};
struct node *start = NULL;

struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node *);
struct node *insert_after(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_before(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);

int main()
{
        int option;
        do
        {
                printf("\n\n *****MAIN MENU *****");
                printf("\n 1: Create a list");
                printf("\n 2: Display the list");
                printf("\n 3: Add a node at the beginning");
                printf("\n 4: Add a node at the end");
                printf("\n 5: Add a node before a given node");
                printf("\n 6: Add a node after a given node");
                printf("\n 7: Delete a node from the beginning");
                printf("\n 8: Delete a node from the end");
                printf("\n 9: Delete a node before a given node");
                printf("\n 10: Delete a node after a given node");
                printf("\n 11: Delete the entire list");
                printf("\n 12: EXIT");
                printf("\n\n Enter your option : ");
                scanf("%d", &option);
                switch(option)
                {
                        case 1: start = create_ll(start);
                                printf("\n DOUBLY LINKED LIST CREATED");
                                break;
```

```
                        case 2: start = display(start);
                                break;
                        case 3: start = insert_beg(start);
                                break;
                        case 4: start = insert_end(start);
                                break;
                        case 5: start = insert_before(start);
                                break;
                        case 6: start = insert_after(start);
                                break;
                        case 7: start = delete_beg(start);
                                break;
                        case 8: start = delete_end(start);
                                break;
                        case 9: start = delete_before(start);
                                break;
                        case 10: start = delete_after(start);
                                break;
                        case 11: start = delete_list(start);
                                printf("\n DOUBLY LINKED LIST DELETED");
                                break;
                }
        }while(option != 12);
        getch();
        return 0;
}

struct node *create_ll(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter –1 to end");
        printf("\n Enter the data : ");
        scanf("%d", &num);
        while(num != –1)
        {
                if(start == NULL)
                {
                        new_node = (struct node*)malloc(sizeof(struct node));
                        new_node -> prev = NULL;
                        new_node -> data = num;
                        new_node -> next = NULL;
                        start = new_node;
```

```
                }
                else
                {
                        ptr=start;
                        new_node = (struct node*)malloc(sizeof(struct node));
                        new_node–>data=num;
                        while(ptr–>next!=NULL)
                                ptr = ptr–>next;
                        ptr–>next = new_node;
                        new_node–>prev=ptr;
                        new_node–>next=NULL;
                }
                printf("\n Enter the data : ");
                scanf("%d", &num);
        }
        return start;
}

struct node *display(struct node *start)
{
        struct node *ptr;
        ptr=start;
        while(ptr!=NULL)
        {
                printf("\t %d", ptr -> data);
                ptr = ptr -> next;
        }
        return start;
}

struct node *insert_beg(struct node *start)
{
        struct node *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        start -> prev = new_node;
        new_node -> next = start;
        new_node -> prev = NULL;
        start = new_node;
        return start;
```

```
}

struct node *insert_end(struct node *start)
{
        struct node *ptr, *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr=start;
        while(ptr -> next != NULL)
                ptr = ptr -> next;
        ptr -> next = new_node;
        new_node -> prev = ptr;
        new_node -> next = NULL;
        return start;
}

struct node *insert_before(struct node *start)
{
        struct node *new_node, *ptr;
        int num, val;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        printf("\n Enter the value before which the data has to be inserted : ");
        scanf("%d", &val);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr = start;
        while(ptr -> data != val)
                ptr = ptr -> next;
        new_node -> next = ptr;
        new_node -> prev = ptr-> prev;
        ptr -> prev -> next = new_node;
        ptr -> prev = new_node;
        return start;
}

struct node *insert_after(struct node *start)
{
        struct node *new_node, *ptr;
        int num, val;
```

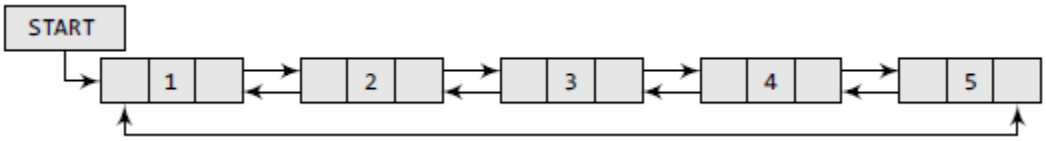
```
        printf("\n Enter the data : ");
        scanf("%d", &num);
        printf("\n Enter the value after which the data has to be inserted : ");
        scanf("%d", &val);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr = start;
        while(ptr -> data != val)
                ptr = ptr -> next;
        new_node -> prev = ptr;
        new_node -> next = ptr -> next;
        ptr -> next -> prev = new_node;
        ptr -> next = new_node;
        return start;
}

struct node *delete_beg(struct node *start)
{
        struct node *ptr;
        ptr = start;
        start = start -> next;
        start -> prev = NULL;
        free(ptr);
        return start;
}

struct node *delete_end(struct node *start)
{
        struct node *ptr;
        ptr = start;
        while(ptr -> next != NULL)
                ptr = ptr -> next;
        ptr -> prev -> next = NULL;
        free(ptr);
        return start;
}

struct node *delete_after(struct node *start)
{
        struct node *ptr, *temp;
        int val;
        printf("\n Enter the value after which the node has to deleted : ");
        scanf("%d", &val);
```

```
        ptr = start;
        while(ptr -> data != val)
                ptr = ptr -> next;
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        return start;
}

struct node *delete_before(struct node *start)
{
        struct node *ptr, *temp;
        int val;
        printf("\n Enter the value before which the node has to deleted : ");
        scanf("%d", &val);
        ptr = start;
        while(ptr -> data != val)
                ptr = ptr -> next;
        temp = ptr -> prev;
        if(temp == start)
                start = delete_beg(start);
        else
        {
                ptr -> prev = temp -> prev;
                temp -> prev -> next = ptr;
        }
        free(temp);
        return start;
}

struct node *delete_list(struct node *start)
{
        while(start != NULL)
        start = delete_beg(start);
        return start;
}
```

Κυκλικές Διπλά Συνδεδεμένες Λίστες

Οι κυκλικές διπλά συνδεδεμένες λίστες είναι ένας συνδυασμός κυκλικών λιστών και διπλά συνδεδεμένων λιστών. Κάθε κόμβος περιλαμβάνει δείκτες προς τον επόμενο και τον προηγούμενο κόμβο ενώ ταυτόχρονα ο τελευταίος κόμβος συνδέεται με τον πρώτο κόμβο της λίστας.

Σχηματικά μια κυκλική διπλά συνδεδεμένη λίστα έχει ως εξής:



START = 1

	DATA	PREV	Next
1	H	9	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	O	7	1

Ακολουθούν οι αλγόριθμοι και ο κώδικας που υλοποιεί τις βασικές λειτουργίες σε μια κυκλική διπλά συνδεδεμένη λίστα.

- **Εισαγωγή κόμβου στην αρχή της λίστας**

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 13
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET PTR = START
Step 6: Repeat Step 7 while PTR -> NEXT != START
Step 7:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 8: SET PTR -> NEXT = NEW_NODE
Step 9: SET NEW_NODE -> PREV = PTR
Step 10: SET NEW_NODE -> NEXT = START
Step 11: SET START -> PREV = NEW_NODE
Step 12: SET START = NEW_NODE
Step 13: EXIT
```

- **Εισαγωγή ενός κόμβου στο τέλος της λίστας**

```
Step 1: IF AVAIL = NULL
            Write OVERFLOW
            Go to Step 12
        [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = START
Step 6: SET PTR = START
Step 7: Repeat Step 8 while PTR -> NEXT != START
Step 8:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: SET NEW_NODE -> PREV = PTR
Step 11: SET START -> PREV = NEW_NODE
Step 12: EXIT
```

- **Διαγραφή του πρώτου κόμβου της λίστας**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4 while PTR -> NEXT != START
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> NEXT = START -> NEXT
Step 6: SET START -> NEXT -> PREV = PTR
Step 7: FREE START
Step 8: SET START = PTR -> NEXT
```

- **Διαγραφή του τελευταίου κόμβου της λίστας**

```
Step 1: IF START = NULL
            Write UNDERFLOW
            Go to Step 8
        [END OF IF]
Step 2: SET PTR = START
Step 3: Repeat Step 4  while PTR -> NEXT != START
Step 4:     SET PTR = PTR -> NEXT
        [END OF LOOP]
Step 5: SET PTR -> PREV -> NEXT = START
Step 6: SET START -> PREV = PTR -> PREV
Step 7: FREE PTR
Step 8: EXIT
```

Code 5

(Υλοποιεί με τη χρήση ξεχωριστών συναρτήσεων όλες τις λειτουργίες σε μια διπλά συνδεδεμένη λίστα)

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
struct node
{
```

```
        struct node *next;
        int data;
        struct node *prev;
};

struct node *start = NULL;
struct node *temp = NULL;

struct node *create_ll(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *);
struct node *delete_list(struct node *);

int main()
{
        int option;
        do
        {
                printf("\n\n *****MAIN MENU *****");
                printf("\n 1: Create a list");
                printf("\n 2: Display the list");
                printf("\n 3: Add a node at the beginning");
                printf("\n 4: Add a node at the end");
                printf("\n 5: Delete a node from the beginning");
                printf("\n 6: Delete a node from the end");
                printf("\n 7: Delete a given node");
                printf("\n 8: Delete the entire list");
                printf("\n 9: EXIT");
                printf("\n\n Enter your option : ");
                scanf("%d", &option);
                switch(option)
                {
                        case 1: start = create_ll(start);
                                printf("\n CIRCULAR DOUBLY LINKED LIST CREATED");
                                break;
                        case 2: start = display(start);
                                break;
                        case 3: start = insert_beg(start);
                                break;
                        case 4: start = insert_end(start);
                                break;
                        case 5: start = delete_beg(start);
                                break;
                        case 6: start = delete_end(start);
                                break;
                        case 7: start = delete_node(start);
                                break;
                        case 8: start = delete_list(start);
```

```
                                printf("\n CIRCULAR DOUBLY LINKED LIST DELETED");
                                break;
                        }
                }while(option != 9);
                getch();
                return 0;
}

struct node *create_ll(struct node *start)
{
                struct node *new_node, *ptr;
                int num;
                printf("\n Enter –1 to end");
                printf("\n Enter the data : ");
                scanf("%d", &num);
                while(num != –1)
                {
                        if(start == NULL)
                        {
                                new_node = (struct node*)malloc(sizeof(struct node));
                                new_node -> prev = NULL;
                                new_node -> data = num;
                                new_node -> next = start;
                                start = new_node;
                        }
                        else
                        {
                                new_node = (struct node*)malloc(sizeof(struct node));
                                new_node -> data = num;
                                ptr = start;
                                while(ptr -> next != start)
                                        ptr = ptr -> next;
                                new_node -> prev = ptr;
                                ptr -> next = new_node;
                                new_node -> next = start;
                                start -> prev = new_node;
                        }
                        printf("\n Enter the data : ");
                        scanf("%d", &num);
                }
                return start;
}

struct node *display(struct node *start)
{
                struct node *ptr;
                ptr = start;
                while(ptr -> next != start)
                {
                        printf("\t %d", ptr -> data);
                        ptr = ptr -> next;
                }
```

```
        printf("\t %d", ptr -> data);
        return start;
}

struct node *insert_beg(struct node *start)
{
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node-> data = num;
        ptr = start;
        while(ptr -> next != start)
                ptr = ptr -> next;
        new_node -> prev = ptr;
        ptr -> next = new_node;
        new_node -> next = start;
        start -> prev = new_node;
        start = new_node;
        return start;
}

struct node *insert_end(struct node *start)
{
        struct node *ptr, *new_node;
        int num;
        printf("\n Enter the data : ");
        scanf("%d", &num);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node -> data = num;
        ptr = start;
        while(ptr -> next != start)
                ptr = ptr -> next;
        ptr -> next = new_node;
        new_node -> prev = ptr;
        new_node -> next = start;
        start-> prev = new_node;
        return start;
}

struct node *delete_beg(struct node *start)
{
        struct node *ptr;
        ptr = start;
        while(ptr -> next != start)
                ptr = ptr -> next;
        ptr -> next = start -> next;
        temp = start;
        start=start–>next;
        start–>prev=ptr;
        free(temp);
        return start;
```

```
}

struct node *delete_end(struct node *start)
{
        struct node *ptr;
        ptr=start;
        while(ptr -> next != start)
                ptr = ptr -> next;
        ptr -> prev -> next = start;
        start -> prev = ptr -> prev;
        free(ptr);
        return start;
}

struct node *delete_node(struct node *start)
{
        struct node *ptr;
        int val;
        printf("\n Enter the value of the node which has to be deleted : ");
        scanf("%d", &val);
        ptr = start;
        if(ptr -> data == val)
        {
                start = delete_beg(start);
                return start;
        }
        else
        {
                while(ptr -> data != val)
                        ptr = ptr -> next;
                ptr -> prev -> next = ptr -> next;
                ptr -> next -> prev = ptr -> prev;
                free(ptr);
                return start;
        }
}

struct node *delete_list(struct node *start)
{
        struct node *ptr;
        ptr = start;
        while(ptr -> next != start)
                start = delete_end(start);
        free(start);
        return start;
}
```