DOUBLY LINKED LISTS

The circular lists have advantages over the linear lists. However, you can only traverse the circular list in one (i.e. forward) direction, which means that you cannot traverse the circular list in backward direction.

This problem can be overcome by using doubly linked lists where there three fields in the node structure:
- `info` field
- `left` pointer
- `right` pointer

The doubly linked lists can be linear or circular.

Each node in doubly linked list can be declared by:

```c
struct node{
    int info;
    struct node *left, *right;
};
typedef struct node nodeptr;
```

A *Linear* Doubly Linked List

A *Circular* Doubly Linked List without Header Node

A *Circular* Doubly Linked List with a *Header Node*
**Primitive Functions in Doubly Circular Lists**

**Delete a node:** The following function can be used to delete a node pointed by a node pointer can be deleted in a doubly circular linked list.

```c
void delete(nodeptr *p, int *px)
{
    nodeptr *l,*r;
    if(p == NULL){
        printf("void deletion\n");
        return;
    }
    *px = p->info; /*the data of the deleted node*/
    l = p->left;
    r = p->right;
    l->right = r;
    r->left = l;
    freenode(p);
}
```

**Insert a node:** The following function `insertright` inserts a node with information field x to the right of a node pointed by p in a doubly circular linked list. `Insertleft` function can be written similarly.

```c
void insertright(nodeptr *p, int x)
{
    nodeptr *q,*r;
    if(p == NULL){
        printf("void insertion\n");
        return;
    }
    q = getnode();
    q->info = x;
    r = p->right;
    r->left = q;
    q->right = r;
    q->left = p;
    p->right = q
}
```

**Ex 1:** A circular double-linked list with header node is maintained to store circles in 2-D. Each circle is defined by three floating point numbers, which are the coordinates of the center, (x,y) and radius r.

a) Define a node structure for this list. (Hint: define a new type called CIRCLE with three float fields and use it as the info field of the node structure).

```c
typedef struct{
    float x,
    float y,
    float r,
} CIRCLE;
```
struct node{
    int flag;  /* 1 for the header 0 otherwise*/
    CIRCLE info;
    struct node *left, *right;
};
typedef struct node nodeptr;

b) Write a function to find the number of circles whose area is less than a given reference area value. Use the following prototype for such a function:

    int NofCircleArea(nodeptr *list, float refArea);

Assume that the header node contains a flag which is 1 for the header node and 0 otherwise.

    int NofCircleArea(nodeptr *list, float refArea)
    {
        nodeptr *p;
        int count =0;
        float a;
        p=list;
        if((p == NULL)){
            printf("Empty list\n");
            exit(1);
        }
        do{
            a = 3.14 * (p->info.r) * (p->info.r);/*area calculated*/
            if( a <= refArea)
                count ++;
            p =p->right;
        }while(p->flag !=1);
        return count;
    }

Ex 2: Consider a doubly circular linked list, where each node contains the name of a company, the budget of the company as well as the number of employees (workers) in that company.

    a) Write an appropriate node structure definition for the records of the doubly circular linked list.

        typedef struct{
            char name[20];
            float budget;
            int employee;
        }COMPANY;

        struct node{
            COMPANY info;
            struct node *left;
            struct node *right;
        };
        typedef struct node nodeptr;
b) Assume that the doubly circular linked list contains only a single node which is the header node with a sentinel -1 as the number of employees.

Given an array where the records of different companies are stored as elements of the array. Write the definition of a function where the elements of the array are copied/inserted into the doubly linked list in ascending order according to the budget of each company.

```c
void CopyArray(nodeptr *dlist, COMPANY *aptr, int size)
{
    int i;
    nodeptr *p=dlist;
    SORT(aptr,size); /*this function will sort the input array*/
    for(i=0;i<size;i++){
        insertright(p,* (aptr+i));
        p = p->right;
    }
}

void SORT(COMPANY *aptr, int size)
{
    int i,j;
    COMPANY hold;
    for(i=0;i<= size-1;i++){
        for(j=0;j<= size-2;j++){
            if( (aptr+j)->budget > (aptr+j+1)->budget ){
                hold = *(aptr+j);
                *(aptr+j) = *(aptr+j+1);
                *(aptr+j+1) = hold;
            }
        }
    }
}

void insertright(nodeptr *p, COMPANY x)
{
    nodeptr *q,*r;
    if(p == NULL){
        printf("void insertion\n");
        return;
    }
    q = getnode(); /* assumed to be allocating necessary node space */
    q->info = x;
    r = p->right;
    r->left = q;
    q->right = r;
    q->left = p;
    p->right = q
}
c) Given a sorted doubly linked list according to the choice (b). Write a function to display the names of the companies with minimum and maximum budgets.

```c
void MinMaxBudget(nodeptr *dlist)
{
    nodeptr *rp, *lp;
    rp = dlist->right; /* sorted list, therefore rp points the minimum */
    lp = dlist->left; /* sorted list, therefore lp points the maximum */
    printf("Company info with Minimum Budget\n");
    printf("Name:%s\nBudget:%.2f\nEmployee:%d\n",rp->info.name,
           rp->info.budget, rp->info.employee);
    printf("Company info with Maximum Budget\n");
    printf("Name:%s\nBudget:%.2f\nEmployee:%d\n",lp->info.name,
           lp->info.budget, lp->info.employee);
}
```