

//Doubly Circular Linked List

```
# include <stdio.h>
# include <conio.h>

struct emp
{
    int code;
    char name[10];
    struct emp *next, *prev;
};

typedef struct emp empl;

empl *start;

void main()
{
    int choice;
    void insert_first();
    void insert_last();
    void insert_specific();
    void delete_first();
    void delete_last();
    void delete_specific_value();
    void delete_specific_nodeno();
    void display();
    void search();
    void sort();

    do
    {
        clrscr();

        printf("\n\t1. Insert First");
        printf("\n\t2. Insert Last");
        printf("\n\t3. Insert Specific");
        printf("\n\t4. Delete First");
        printf("\n\t5. Delete Last");
        printf("\n\t6. Delete Specific by Value");
        printf("\n\t7. Delete Specific by Node No");
        printf("\n\t8. Display");
        printf("\n\t9. Search");
        printf("\n\t10. Sort");
        printf("\n\t0. Exit");

        printf("\n\tEnter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
                insert_first();
                break;
            case 2:
```

```

    insert_last();
    break;
case 3:
    insert_specific();
    break;
case 4:
    delete_first();
    break;
case 5:
    delete_last();
    break;
case 6:
    delete_specific_value();
    break;
case 7:
    delete_specific_nodeno();
    break;
case 8:
    display();
    break;
case 9:
    search();
    break;
case 10:
    sort();
    break;
case 0:
    printf("\n\tEnd of the program");
    break;
default:
    printf("\n\tInvalid Choice");
    break;
}
getch();
}
while(choice != 0);
}

```

```

void insert_first()
{
    struct emp *newnode,*temp;
    if(start == NULL)
    {
        start = (struct emp *) malloc(sizeof(struct emp));
        start->next = start;
        start->prev = start;
        newnode = start;
    }
    else
    {
        newnode = (struct emp *) malloc(sizeof(struct emp));
        newnode->next = start;
        newnode->prev = start->prev;
        start->prev = newnode;
    }
}

```

```

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

temp->next = newnode;

start = newnode;
}

printf("\n\tEnter Emp. Code = ");
scanf("%d",&newnode->code);

printf("\n\tEnter Emp. Name = ");
fflush(stdin);
gets(newnode->name);
}

void insert_last()
{
    struct emp *newnode, *temp;

    if(start == NULL)
    {
        start = (struct emp *) malloc(sizeof(struct emp));
        start->next = start;
        start->prev = start;
        newnode = start;
    }
    else
    {
        temp = start;
        while(temp->next != start)
        {
            temp = temp->next;
        }
        newnode = (struct emp *) malloc(sizeof(struct emp));
        newnode->next = temp->next;
        newnode->prev = temp;
        temp->next = newnode;
        start->prev = newnode;
    }

    printf("\n\tEnter Emp. Code = ");
    scanf("%d",&newnode->code);

    printf("\n\tEnter Emp. Name = ");
    fflush(stdin);
    gets(newnode->name);
}

void insert_specific()
{
    struct emp *newnode, *temp;

```

```

int a, nodeno, count=1;

if(start == NULL)
{
    insert_first();
}
else
{
    temp = start;
    while(temp->next != start)
    {
        temp = temp->next;
        count++;
    }

    do
    {
        printf("\n\tEnter node no between 1 to %d = ", count+1);
        scanf("%d",&nodeno);
    }
    while(nodeno < 1 || nodeno > count+1);

    if(nodeno == 1)
        insert_first();
    else if(nodeno == count+1)
        insert_last();
    else
    {
        temp = start;
        a=1;
        while(a < nodeno-1)
        {
            a++;
            temp = temp->next;
        }

        newnode = (struct emp *) malloc(sizeof(struct emp));
        newnode->next = temp->next;
        newnode->prev = temp;

        temp->next->prev = newnode;
        temp->next = newnode;

        printf("\n\tEnter Emp. Code = ");
        scanf("%d",&newnode->code);

        printf("\n\tEnter Emp. Name = ");
        fflush(stdin);
        gets(newnode->name);
    }
}

void delete_first()
{

```

```

struct emp *deletenode, *temp;

if(start == NULL)
{
    printf("\n\tDoubly Circular Linked List is Empty");
}
else
{
    deletenode = start;
    temp = start;

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

    if(start->next != start)
    {
        start->next->prev = start->prev;
        start = start->next;
    }
    else
    {
        start = NULL;
    }
    printf("\n\tDelete Node Information : ");
    printf("\n\tEmp. Code = %d",deletenode->code);
    printf("\n\tEmp. Name = %s",deletenode->name);

    free(deletenode);
}
}

```

```

void delete_last()
{
    struct emp *deletenode, *temp;

    if(start == NULL)
    {
        printf("\n\tDoubly Circular Linked List is Empty");
    }
    else
    {
        if(start->next == start)
        {
            deletenode = start;
            start = NULL;
        }
        else
        {
            temp = start;
            while(temp->next->next != start)
            {
                temp = temp->next;
            }

```

```

}
deletenode = temp->next;
temp->next = temp->next->next;
start->prev = temp;
}

printf("\n\tDelete Node Information = ");
printf("\n\tEmp. Code = %d",deletenode->code);
printf("\n\tEmp. Name = %s",deletenode->name);
free(deletenode);
}
}

```

```

void delete_specific_value()
{
    struct emp *deletenode=NULL, *temp;
    int ecode;

    if(start == NULL)
    {
        printf("\n\tDoubly Circular Linked List is Empty");
    }
    else
    {
        printf("\n\tEnter Emp. Code to Delete : ");
        scanf("%d",&ecode);

        if(start->code == ecode)
        {
            delete_first();
        }
        else
        {
            temp = start;

            while(temp->next != start)
            {
                if(temp->next->code == ecode)
                {
                    deletenode = temp->next;
                    deletenode->next->prev = temp;
                    temp->next = temp->next->next;
                    break;
                }
                temp = temp->next;
            }

            if(deletenode == NULL)
            {
                printf("\n\tEmp. Code %d not found", ecode);
            }
            else
            {
                printf("\n\tDelete Node Information =");
                printf("\n\tEmp. Code = %d",deletenode->code);
            }
        }
    }
}

```

```

    printf("\n\tEmp. Name = %s",deletenode->name);
    free(deletenode);
}
}
}
}

```

```

void delete_specific_nodeno()
{
    struct emp *deletenode, *temp;
    int a, count = 1, nodeno;

    if(start == NULL)
    {
        printf("\n\tDoubly Circular Linked List is Empty");
    }
    else
    {
        temp = start;
        while(temp->next != start)
        {
            temp = temp->next;
            count++;
        }

        do
        {
            printf("\n\tEnter Node no. to delete between 1 to %d = ", count);
            scanf("%d",&nodeno);
        }
        while(nodeno < 1 || nodeno > count);

        if(nodeno == 1)
        {
            delete_first();
        }
        else if(nodeno == count)
        {
            delete_last();
        }
        else
        {
            temp = start;
            a=1;

            while(a < nodeno-1)
            {
                temp = temp->next;
                a++;
            }

            deletenode = temp->next;

            deletenode->next->prev = deletenode->prev;
            temp->next = temp->next->next;

```

```

printf("\n\tDelete Node Information = ");
printf("\n\tEmp. Code = %d",deletenode->code);
printf("\n\tEmp. Name = %s",deletenode->name);

free(deletenode);
}
}
}

```

```

void display()
{
empl *temp;
if(start == NULL)
{
printf("\n\tDoubly Circular Linked List is Empty");
}
else
{
temp = start;

printf("\n\tForward Direction");
printf("\n\tEmp. Code\t\tEmp. Name");
while(temp->next != start)
{
printf("\n\t%d\t\t\t%s",temp->code, temp->name);
temp = temp->next;
}
printf("\n\t%d\t\t\t%s",temp->code, temp->name);

printf("\n\n\tBackward Direction");
printf("\n\tEmp. Code\t\tEmp. Name");
while(temp->prev != start->prev)
{
printf("\n\t%d\t\t\t%s",temp->code, temp->name);
temp = temp->prev;
}
printf("\n\t%d\t\t\t%s",temp->code, temp->name);
}
}

```

```

void search()
{
int empcode, flag = 0;
struct emp *temp;

if(start == NULL)
{
printf("\n\tDoubly Circular Linked List is Empty");
}
else
{
printf("\n\tEnter Emp. Code to Search : ");
scanf("%d",&empcode);

```



```

if(start->code == empcode)
{
    //flag = 1;
    temp = start;
}
else
{
    temp = start;

    while(temp->next != start)
    {
        if(temp->next->code == empcode)
        {
            temp = temp->next;
            flag = 1;
            break;
        }
        temp = temp->next;
    }
}
if(flag == 0)
{
    printf("\n\tEmp. Code %d not found", empcode);
}
else
{
    printf("\n\tEmp. Code Found");
    printf("\n\tEmp. Code = %d",temp->code);
    printf("\n\tEmp. Name = %s",temp->name);
}
}
}

```

```

void sort()
{
    empl *temp1, *temp2;
    int ecode;
    char ename[10];

    if(start == NULL)
    {
        printf("\n\tDoubly Circular Linked List is Empty");
    }
    else
    {
        temp1 = start;
        while(temp1->next != start)
        {
            temp2 = temp1->next;
            while(temp2 != start)
            {
                if(temp1->code > temp2->code)
                {
                    ecode = temp1->code;

```

```
temp1->code = temp2->code;
temp2->code = ecode;

strcpy(ename, temp1->name);
strcpy(temp1->name, temp2->name);
strcpy(temp2->name, ename);
}
temp2 = temp2->next;
}
temp1 = temp1->next;
}
display();
}
}
```