2347214
Aravind Nandakumar
II MCA - B
Christ University

Lab Exercise 2

Data Structures Using C

Implement linked list and its operations

Consider each node as a structure representation of data for your domain.
Perform all operations and implement different types of linked list

Singly Linked List

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

struct worker{
    char name[10];
    struct worker *right;
}*newptr,*first,*last,*temp,*prev,*next;

int numberOfNodes=0;


int create(){
    char ch;
    while(1)
    {
        newptr=(struct worker*) malloc(sizeof(struct worker));
        if(newptr==NULL){
                printf("Memory allocation error");
                return 0;
            }
        printf("\nEnter Name of worker");
        scanf("%s",&newptr->name);
```

```
        numberOfNodes++;
        newptr->right=NULL;
        if(first==NULL)
           first=temp=newptr;
           else
            {
                temp->right=newptr;
                temp=temp->right;
            }
        printf("want to add more workers(Y/N)");
        ch=getch();
        if(ch=='n'||ch=='N')
         return(0);
    temp=first;
    while(temp->right!=NULL)
    {
        temp=temp->right;
        last=temp;
    }
  }
}


void display(){
    temp= first;
    if(temp==NULL){
        printf("There are no workers\n");
        return;
    }
    while(temp != NULL) {
        printf("[%s]--->",temp -> name );
        temp = temp-> right ;
    }
    printf("NULL \n");
}


void insert_begining(){
    newptr = (struct worker *)malloc( sizeof( struct worker ) );
    if(newptr==NULL){
```

```
        printf("Memory allocation error");
        return;
    }
    printf("\nEnter Name of new Worker : ");
    scanf("%s",&newptr->name);
    numberOfNodes++;
    newptr->right=NULL;
    if(first == NULL)
    {
        first=last=newptr;
    }
    else
    {
        newptr->right=first;
        first=newptr;
    }
}

void insert_end(){
    newptr=(struct worker*)malloc(sizeof(struct worker));
    if(newptr==NULL){
        printf("Memory allocation error");
        return;
    }
    printf("\nEnter Name of new Worker : ");
    scanf("%s",&newptr->name);
    numberOfNodes++;
    newptr->right=NULL;
    temp= first;
    while(temp!=NULL){
        last=temp;
        temp=temp->right;
    }
    last->right=newptr;
    newptr->right=NULL;
}

void insert_middle(){
    int pos,c;c=0;
    newptr=(struct worker*)malloc(sizeof(struct worker));
```

```
    if(newptr==NULL){
        printf("Memory allocation error");
        return ;
    }
    printf("Enter Postion for worker to be inserted : ");
    scanf("%d",&pos);
    printf("\nEnter Name of new Worker : ");
    scanf("%s",&newptr->name);
    numberOfNodes++;
    temp= first;
    while (temp!=NULL)
    {
        c++;
        if(c==pos-1){
           next=temp->right;
           newptr->right=next;
           temp->right=newptr;
           break;
        }
        temp=temp->right;
    }
}

void delete_begining(){
    if(first==NULL){
        printf("\nThere are no Workers");
    }
    else{
        temp = first;
        first = first -> right;
        free(temp);
        numberOfNodes--;
        printf("\nFirst Worker deleted\n");


    }
}

void delete_end(){
    if(first==NULL){
```

```
        printf("\nThere are no Workers");
        return;
    }
    temp=first;
    while(temp->right!=NULL)
    {
        prev=temp;
        temp=temp->right;
        last=temp;
    }
    prev->right=NULL;
    last=prev;
    printf("\nLast Worker deleted\n");
    numberOfNodes--;
    free(temp);
}

void delete_middle(){
    if(first==NULL){
        printf("\nThere are no workers");
    }
    else{
        int pos, c; c = 0;
        printf("Enter the position of the worker you want to delete: ");
        scanf("%d", &pos);

        temp = first;
        while(temp->right != NULL){
            c++;
            if(c == pos - 1){
                prev = temp;
                next = temp->right->right;
                free(temp->right);
                prev->right = next;
                printf("\nWorker at position %d deleted\n", pos);
                numberOfNodes--;
                break;
            }
            temp = temp->right;
        }
```

```
    }
}

void search(){
    char search_name[25];
    int pos ,foundFlag=0;pos=0;
    temp = first;
    printf("\nEnter name of the worker you want to find : ");
    scanf("%s",&search_name);
    while(temp!=NULL){
        pos++;
        if(strcmpi(search_name,temp->name)==0){
            foundFlag =1;
            printf("\nWorker Found at %d",pos);
        }
        temp= temp->right;
    }
    if(foundFlag==0){
        printf("\n\tNo such worker found!");
    }
}

void exit_program(){
    temp = first;
    while (temp != NULL) {
        struct worker* nextNode = temp->right;
        free(temp);
        temp = nextNode;
    }
    exit(0);
}


void main()
{
    int opt;
    opt=0;
    first=temp=NULL;
    while(1)
 {
```

```
    printf("\n");
    printf(" +---------Worker-Menu---------+\n");
    printf(" | 1.Create Workers             |\n");
    printf(" | 2.Display Workers            |\n");
    printf(" | 3.Insert Worker Begining     |\n");
    printf(" | 4.Insert Worker Middle       |\n");
    printf(" | 5.Insert Worker End          |\n");
    printf(" | 6.Delete Worker Begining     |\n");
    printf(" | 7.Delete Worker Middle       |\n");
    printf(" | 8.Delete Worker End          |\n");
    printf(" | 9.Search Worker              |\n");
    printf(" | 10.Exit                      |\n");
    printf(" +------------------------------+\n");
    printf("enter your option");
    scanf("%d",&opt);
    switch(opt)
    {
        case 1:create();break;
        case 2:display();break;
        case 3:insert_begining();break;
        case 4:numberOfNodes>=2?insert_middle():printf("\nNeed More Than 1
Node\n");break;
        case 5:insert_end();break;
        case 6:delete_begining();break;
        case 7:numberOfNodes>=2?delete_middle():printf("\nNeed More Than 1
Node\n");break;
        case 8:delete_end();break;
        case 9:search();break;
        case 10:exit_program();
    }
    getch();
 }
}
```

Doubly Linked List

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct worker{
    char name[10];
    struct worker *right;
    struct worker *left;
}*temp,*prev,*first,*last,*newptr,*next;

int numOfNodes = 0;

int create()
{
    char ch;
    while(1)
    {
        newptr=(struct worker*) malloc(sizeof(struct worker));
        if(newptr==NULL){
            printf("Memory allocation error");
            return 0;
        }
        printf("\nEnter Name of worker : ");
        scanf("%s",&newptr->name);
        numOfNodes++;
        newptr->right=NULL;
        newptr->left=NULL;
        if(first==NULL)
           first=temp=newptr;
           else
            {
                temp->right=newptr;
                newptr->left=temp;
                temp=temp->right;
            }
        printf("Want to add more Workers(Y/N) : ");
        ch=getch();
        if(ch=='n'||ch=='N')
```

```
         return(0);
    temp=first;
    while(temp->right!=NULL)
    {
        temp=temp->right;
        last=temp;
    }
   }


}
void display_forward()
{
    if(first==NULL){
        printf("There are no workers\n");
        return;
    }
    temp=first;
    printf("Forward Display of Workers : \n");
    printf("NULL");
    while(temp!=NULL)
    {
        printf("<--[%s]-->",temp->name);
        temp=temp->right;
    }
    printf("NULL\n");
}

void display_backward()
{
    if(first==NULL){
        printf("There are no workers\n");
        return;
    }
    temp=first;
    printf("Reverse Display of Workers : \n");
    while(temp->right!=NULL)
    {
        temp=temp->right;
        last=temp;
```

```
    }
    temp=last;
    printf("NULL");
    while(temp!=NULL)
    {
        printf("<--[%s]-->",temp->name);
        temp=temp->left;
    }
    // printf("<--[%s]-->",first->name);
    printf("NULL\n");
}

void search(){
    if(first==NULL){
        printf("There are no workers\n");return;
    }
    char search_name[10];
    printf("Enter Worker Name to be Searched : ");
    scanf("%s",&search_name);
    temp = first;
    int pos = 0;
    int foundFlag=0;
        while(temp->right!=NULL)
       {
        pos++;
        if(strcmpi(search_name,temp->name)==0)
        {

            printf("Worker found at position : %d ",pos);
            foundFlag=1;
            break;
        }
        temp=temp->right;
    }
    if(foundFlag==0){
        printf("Worker is not in List");
    }


}
```

```
void insert_begining()
{
   newptr=(struct worker*) malloc(sizeof(struct worker));
   if(newptr==NULL){
        printf("Memory allocation error");
        return 0;
    }
   printf("\nEnter Worker Name ");
   scanf("%s",&newptr->name);
   numOfNodes++;
   newptr->left=NULL;
   first->left=newptr;
   newptr->right=first;
   first=newptr;
   printf("\nWorker Added At Begining");
}

void insert_end()
{
   newptr=(struct worker*) malloc(sizeof(struct worker));
   if(newptr==NULL){
        printf("Memory allocation error");
        return 0;
    }
   printf("\nEnter Worker Name : ");
   scanf("%s",&newptr->name);
   numOfNodes++;
    newptr->right=NULL;
    last->right=newptr;
    newptr->left=last;
    last=newptr;
    printf("\nWorker Added At end");
}

void delete_begining()
{
    if(first==NULL){
        printf("There are no workers\n");return;
    }
```

```
    temp=first;
    first=first->right;
    first->left=NULL;
    temp->right=NULL;
    free(temp);
    numOfNodes--;
    printf("\nWorker Deleted from Begining");
}

void delete_end()
{

    if(first==NULL){
        printf("There are no workers\n");return;
    }
    temp=last;
    last=last->left;
    last->right=NULL;
    temp->left=NULL;
    numOfNodes--;
    free(temp);
    printf("\nWorker Deleted from end");

}


void delete_middle(){
    if(first==NULL){
        printf("\nThere are no workers");
    }
    else{
        int pos, c; c = 0;
        printf("Enter the position of the worker you want to delete : ");
        scanf("%d", &pos);

        temp = first;
        while(temp->right != NULL){
            c++;
            if(c == pos){
                prev= temp->left;
```

```
                next = temp->right;
                prev->right = next;
                next->left = prev;
                temp->left=NULL;
                temp->right=NULL;
                free(temp);
                printf("\nWorker at position %d deleted\n", pos);
                numOfNodes--;
                break;
            }
            temp = temp->right;
        }
    }
}


void insert_middle()
{
    int pos,c;c=0;
    newptr=(struct worker*) malloc(sizeof(struct worker));
    if(newptr==NULL){
        printf("Memory allocation error");
        return 0;
    }
    printf("\nEnter the position  at which insert Worker : ");
    scanf("%d",&pos);
    printf("\nEnter Worker Name : ");
    scanf("%s",&newptr->name);
    numOfNodes++;
    temp=first;
    while(temp->right!=NULL)
    {
        c++;
        if(c==pos)
        {
            prev=temp->left;
            prev->right=newptr;
            newptr->left=prev;
            temp->left=newptr;
            newptr->right=temp;
```

```
        }
        temp=temp->right;
    }

}

void exit_program(){
    temp = first;
    while (temp != NULL) {
        struct worker* nextNode = temp->right;
        free(temp);
        temp = nextNode;
    }
    exit(0);
}

void main()
{
    int opt;
    opt=0;
    first=temp=NULL;
    while(1)
 {
    printf("\n");
    printf(" +---------Worker-Menu---------+\n");
    printf(" | 1.Create Workers            |\n");
    printf(" | 2.Display Workers           |\n");
    printf(" | 3.Display Workers Reverse   |\n");
    printf(" | 4.Insert Worker Begining    |\n");
    printf(" | 5.Insert Worker Middle      |\n");
    printf(" | 6.Insert Worker End         |\n");
    printf(" | 7.Delete Worker Begining    |\n");
    printf(" | 8.Delete Worker Middle      |\n");
    printf(" | 9.Delete Worker End         |\n");
    printf(" | 10.Search Worker            |\n");
    printf(" | 11.Exit                     |\n");
    printf(" +-----------------------------+\n");
    printf("Enter your option");
    scanf("%d",&opt);
    switch(opt)
```

```
    {
        case 1:create();break;
        case 2:display_forward();break;
        case 3:display_backward();break;
        case 4:insert_begining();break;
        case 5:numOfNodes>=2?insert_middle():printf("\nNeed More Than 1
Node\n");break;
        case 6:insert_end();break;
        case 7:delete_begining();break;
        case 8:numOfNodes>=2?delete_middle():printf("\nNeed More Than 1
Node\n");break;
        case 9:delete_end();break;
        case 10:search();break;
        case 11:exit_program();
    }
    getch();
 }
}
```

Singly Circular Linked List

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

struct worker{
    char name[10];
    struct worker *right;
}*newptr,*first,*last,*temp,*prev,*next;


int create(){
    char ch;
    while(1)
    {
        newptr=(struct worker*) malloc(sizeof(struct worker));
        if(newptr==NULL){
```

```
            printf("Memory allocation error");
            return 0;
        }
        printf("\nEnter Name of worker");
        scanf("%s",&newptr->name);
        newptr->right=NULL;
        if(first==NULL)
           first=temp=last=newptr;
           else
            {
                temp->right=newptr;
                temp=temp->right;
            }
        printf("want to add more workers(Y/N)");
        ch=getch();
        if(ch=='n'||ch=='N')
         {
            temp=first;
            while(temp->right!=NULL)
            {
                temp=temp->right;
                last=temp;
            }
            last->right= first;
            return(0);
         }
  }

}


void display(){
    temp= first;
    if(temp==NULL){
        printf("There are no workers\n");
        return;
    }
    do{
        printf("[%s]--->",temp -> name );
        temp = temp-> right ;
```

```
    }while(temp!=first);
    printf("(%s)",last->right->name);
}

void insert_begining(){
    newptr = (struct worker *)malloc( sizeof( struct worker ) );
    if(newptr==NULL){
            printf("Memory allocation error");
            return;
        }
    printf("\nEnter Name of new Worker : ");
    scanf("%s",&newptr->name);
    newptr->right=NULL;
    if(first == NULL)
    {
        first=last=newptr;
    }
    else
    {
        newptr->right=first;
        first=newptr;
    }
    last->right= first;
}

void insert_end(){
    newptr=(struct worker*)malloc(sizeof(struct worker));
    if(newptr==NULL){
            printf("Memory allocation error");
            return;
        }
    printf("\nEnter Name of new Worker : ");
    scanf("%s",&newptr->name);
    newptr->right=NULL;
    last->right=newptr;
    last=newptr;
    last->right=first;

}
```

```
void insert_middle(){
    int pos,c;c=0;
    newptr=(struct worker*)malloc(sizeof(struct worker));
    printf("Enter Postion for worker to be inserted : ");
    scanf("%d",&pos);
    printf("\nEnter Name of new Worker : ");
    scanf("%s",&newptr->name);
    temp= first;
    while (temp!=NULL)
    {
        c++;
        if(c==pos-1){
           next=temp->right;
           newptr->right=next;
           temp->right=newptr;
           break;
        }
        temp=temp->right;
    }
}

void delete_begining(){
    if(first==NULL){
        printf("\nThere are no Workers");
    }
    else{
        temp = first;
        first = first -> right;
        last->right=first;
        free(temp);
        printf("\nFirst Worker deleted\n");


    }
}

void delete_end(){
    if(first==NULL){
        printf("\nThere are no Workers");
        return;
```

```
    }
    temp=first;
    while(temp!=last){
        prev=temp;
        temp=temp->right;
    }
    prev->right=NULL;
    last=prev;
    last->right=first;
    printf("\nLast Worker deleted\n");
    free(temp);
}

void delete_middle(){
    if(first==NULL){
        printf("\nThere are no workers");
    }
    else{
        int pos, c; c = 0;
        printf("Enter the position of the worker you want to delete: ");
        scanf("%d", &pos);

        temp = first;
        while(temp->right != NULL){
            c++;
            if(c == pos - 1){
                prev = temp;
                next = temp->right->right;
                free(temp->right);
                prev->right = next;
                printf("\nWorker at position %d deleted\n", pos);
                break;
            }
            temp = temp->right;
        }
    }
}

void search(){
    char search_name[25];
```

```
    int pos ,foundFlag=0;pos=0;
    temp = first;
    printf("\nEnter name of the worker you want to find : ");
    scanf("%s",&search_name);

    do{
        pos++;
        if(strcmpi(search_name,temp->name)==0){
            foundFlag =1;
            printf("\nWorker Found at %d",pos);
            break;
        }
        temp= temp->right;

    }while(temp!=first);

    if(foundFlag==0){
        printf("\n\tNo such worker found!");
    }
}

void exit_program(){
    temp = first;
    while (temp != NULL) {
        struct worker* nextNode = temp->right;
        free(temp);
        temp = nextNode;
    }
    exit(0);
}


void main()
{
    int opt;
    opt=0;
    first=temp=NULL;
    while(1)
 {
    printf("\n");
```

```
    printf(" +---------Worker-Menu---------+\n");
    printf(" | 1.Create Workers             |\n");
    printf(" | 2.Display Workers            |\n");
    printf(" | 3.Insert Worker Begining     |\n");
    printf(" | 4.Insert Worker Middle       |\n");
    printf(" | 5.Insert Worker End          |\n");
    printf(" | 6.Delete Worker Begining     |\n");
    printf(" | 7.Delete Worker Middle       |\n");
    printf(" | 8.Delete Worker End          |\n");
    printf(" | 9.Search Worker              |\n");
    printf(" | 10.Exit                      |\n");
    printf(" +------------------------------+\n");
    printf("enter your option");
    scanf("%d",&opt);
    switch(opt)
    {
        case 1:create();break;
        case 2:display();break;
        case 3:insert_begining();break;
        case 4:insert_middle();break;
        case 5:insert_end();break;
        case 6:delete_begining();break;
        case 7:delete_middle();break;
        case 8:delete_end();break;
        case 9:search();break;
        case 10:exit_program();
    }
    getch();
 }
}
```

Doubly Circular Linked List

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct worker{
    char name[10];
    struct worker *right;
    struct worker *left;
}*temp,*prev,*first,*last,*newptr,*next;

int create()
{
    char ch;
    while(1)
    {
        newptr=(struct worker*) malloc(sizeof(struct worker));
        if(newptr==NULL){
                printf("Memory allocation error");
                return 0;
        }
        printf("\nEnter Name of worker : ");
        scanf("%s",&newptr->name);
        newptr->right=NULL;
        newptr->left=NULL;
        if(first==NULL)
           first=temp=last=newptr;
        else
            {
                temp->right=newptr;
                newptr->left=temp;
                temp=temp->right;
            }

        printf("Want to add more Workers(Y/N)");
        ch=getch();
        if(ch=='n'||ch=='N')
        {
            temp=first;
```

```
            while(temp->right!=NULL)
            {
                temp=temp->right;
                last=temp;
            }
            last->right = first;
            first->left = last;
            return(0);
        }

    }
}


void display_forward()
{
    temp= first;
    if(temp==NULL){
        printf("There are no workers\n");
        return;
    }
    printf("Forward Display of Workers : \n");
    do{
        printf("[%s]--->",temp -> name );
        temp = temp-> right ;
    }while(temp!=first);
    printf("(%s)",last->right->name);
}

void display_backward()
{
    temp= last;
    if(temp==NULL){
        printf("There are no workers\n");
        return;
    }
    printf("Backward Display of Workers : \n");
    do{
        printf("[%s]--->",temp -> name );
        temp = temp-> left ;
```

```
    }while(temp!=last);
    printf("(%s)",first->left->name);
}

void search(){

    char search_name[10];
    printf("Enter Worker Name to be Searched");
    scanf("%s",&search_name);
    temp = first;
    int pos = 0;
    int foundFlag=0;
        while(temp->right!=NULL)
       {
        pos++;
        if(strcmpi(search_name,temp->name)==0)
        {

            printf("Worker found at position : %d ",pos);
            foundFlag=1;
            break;
        }
        temp=temp->right;
    }
    if(foundFlag==0){
        printf("Worker is not in List");
    }


}

void insert_begining()
{
   newptr=(struct worker*) malloc(sizeof(struct worker));
   if(newptr==NULL){
        printf("Memory allocation error");
        return;
    }
   printf("\nEnter Worker Name ");
   scanf("%s",&newptr->name);
```

```
   newptr->left=last;
   first->left=newptr;
   newptr->right=first;
   first=newptr;
   last->right=first;

}

void insert_end()
{
   newptr=(struct worker*) malloc(sizeof(struct worker));
   if(newptr==NULL){
        printf("Memory allocation error");
        return;
    }
   printf("\nEnter Worker Name ");
   scanf("%s",&newptr->name);
    newptr->right=first;
    last->right=newptr;
    newptr->left=last;
    last=newptr;
    first->left=last;
}

void delete_begining()
{
    temp=first;
    first=first->right;
    first->left=last;
    temp->right=NULL;
    temp->left=NULL;
    last->right=first;
    free(temp);
}

void delete_end()
{
    temp=last;
    last=last->left;
    last->right=first;
```

```
    temp->left=NULL;
    temp->right=NULL;
    first->left=last;
    free(temp);
}

void delete_middle(){
    if(first==NULL){
        printf("\nThere are no workers");
    }
    else{
        int pos, c; c = 0;
        printf("Enter the position of the worker you want to delete: ");
        scanf("%d", &pos);

        temp = first;
        while(temp!= NULL){
            c++;
            if(c == pos){
                prev= temp->left;
                next = temp->right;
                prev->right = next;
                next->left = prev;
                temp->left=NULL;
                temp->right=NULL;
                free(temp);
                printf("\nWorker at position %d deleted\n", pos);
                break;
            }
            temp = temp->right;
        }
    }
}


void insert_middle()
{
    int pos,c;c=0;
    newptr=(struct worker*) malloc(sizeof(struct worker));
    if(newptr==NULL){
```

```
        printf("Memory allocation error");
        return;
    }
    printf("\nEnter the position  at which insert Worker");
    scanf("%d",&pos);
    printf("\nEnter Worker Name");
    scanf("%s",&newptr->name);
    temp=first;
    while(temp!=NULL)
    {
        c++;
        if(c==pos)
        {
            prev=temp->left;
            prev->right=newptr;
            newptr->left=prev;
            temp->left=newptr;
            newptr->right=temp;
            break;
        }
        temp=temp->right;
    }

}
void main()
{
    int opt;
    opt=0;
    first=temp=NULL;
    while(1)
 {
    printf("\n");
    printf(" +---------Worker-Menu---------+\n");
    printf(" | 1.Create Workers            |\n");
    printf(" | 2.Display Workers           |\n");
    printf(" | 3.Display Workers Reverse   |\n");
    printf(" | 4.Insert Worker Begining    |\n");
    printf(" | 5.Insert Worker Middle      |\n");
    printf(" | 6.Insert Worker End         |\n");
    printf(" | 7.Delete Worker Begining    |\n");
```

```
    printf(" | 8.Delete Worker Middle      |\n");
    printf(" | 9.Delete Worker End         |\n");
    printf(" | 10.Search Worker            |\n");
    printf(" | 11.Exit                     |\n");
    printf(" +-----------------------------+\n");
    printf("Enter your option");
    scanf("%d",&opt);
    switch(opt)
    {
        case 1:create();break;
        case 2:display_forward();break;
        case 3:display_backward();break;
        case 4:insert_begining();break;
        case 5:insert_middle();break;
        case 6:insert_end();break;
        case 7:delete_begining();break;
        case 8:delete_middle();break;
        case 9:delete_end();break;
        case 10:search();break;
        case 11:exit(0);
    }
    getch();
 }
}
```

Singly Linked List

```
  +---------Worker-Menu---------+
  | 1.Create Workers            |
  | 2.Display Workers           |
  | 3.Insert Worker Begining    |
  | 4.Insert Worker Middle      |
  | 5.Insert Worker End         |
  | 6.Delete Worker Begining    |
  | 7.Delete Worker Middle      |
  | 8.Delete Worker End         |
  | 9.Search Worker             |
  | 10.Exit                     |
  +-----------------------------+
enter your option
```

```
enter your option1

Enter Name of workerRam
want to add more workers(Y/N)
Enter Name of workerSam
want to add more workers(Y/N)
Enter Name of workerTim
want to add more workers(Y/N)
```

```
enter your option2
[Ram]--->[Sam]--->[Tim]--->NULL
```

```
enter your option3

Enter Name of new Worker : Jim
```

```
enter your option2
[Jim]--->[Ram]--->[Sam]--->[Tim]--->NULL
```

```
enter your option4
Enter Postion for worker to be inserted : 3

Enter Name of new Worker : Manny
```

```
enter your option2
[Jim]--->[Ram]--->[Manny]--->[Sam]--->[Tim]--->NULL
```

```
enter your option5

Enter Name of new Worker : Yohan
```

```
enter your option2
[Jim]--->[Ram]--->[Manny]--->[Sam]--->[Tim]--->[Yohan]--->NULL
```

```
enter your option6

First Worker deleted
```

```
enter your option2
[Ram]--->[Manny]--->[Sam]--->[Tim]--->[Yohan]--->NULL
```

```
enter your option7
Enter the position of the worker you want to delete: 3

Worker at position 3 deleted
```

```
enter your option2
[Ram]--->[Manny]--->[Tim]--->[Yohan]--->NULL
```

```
enter your option8

Last Worker deleted
```

```
enter your option
2
[Ram]--->[Manny]--->[Tim]--->NULL
```

```
enter your option9

Enter name of the worker you want to find : Manny

Worker Found at 2
```

Doubly Linked List

```
 +---------Worker-Menu---------+
 | 1.Create Workers            |
 | 2.Display Workers           |
 | 3.Display Workers Reverse   |
 | 4.Insert Worker Begining    |
 | 5.Insert Worker Middle      |
 | 6.Insert Worker End         |
 | 7.Delete Worker Begining    |
 | 8.Delete Worker Middle      |
 | 9.Delete Worker End         |
 | 10.Search Worker            |
 | 11.Exit                     |
 +-----------------------------+
Enter your option
```

```
Enter your option
1

Enter Name of worker : Ram
Want to add more Workers(Y/N) :
Enter Name of worker : Tim
Want to add more Workers(Y/N) :
Enter Name of worker : Jon
Want to add more Workers(Y/N) :
Enter Name of worker : Timmy
Want to add more Workers(Y/N) :
```

```
Enter your option2
Forward Display of Workers :
NULL<--[Ram]--><--[Tim]--><--[Jon]--><--[Timmy]-->NULL
```

```
Enter your option3
Reverse Display of Workers :
NULL<--[Timmy]--><--[Jon]--><--[Tim]--><--[Ram]-->NULL
```

```
Enter your option8
Enter the position of the worker you want to delete : 3

Worker at position 3 deleted
```

```
Enter your option2
Forward Display of Workers :
NULL<--[Jim]--><--[Ram]--><--[Jon]--><--[Timmy]--><--[Yohan]-->NULL
```

```
Enter your option9

Worker Deleted from end
```

```
Enter your option2
Forward Display of Workers :
NULL<--[Jim]--><--[Ram]--><--[Jon]--><--[Timmy]-->NULL
```

```
Enter your option10
Enter Worker Name to be Searched : Jon
Worker found at position : 3
```

```
Enter your option4

Enter Worker Name Manu

Worker Added At Begining
```

```
Enter your option2
Forward Display of Workers :
NULL<--[Manu]--><--[Ram]--><--[Tim]--><--[Jon]--><--[Timmy]-->NULL
```

```
Enter your option5

Enter the position  at which insert Worker : 2

Enter Worker Name : Jim
```

```
nter your option2
orward Display of Workers :
ULL<--[Manu]--><--[Jim]--><--[Ram]--><--[Tim]--><--[Jon]--><--[Timmy]-->NULL
```

```
Enter your option7

Worker Deleted from Begining
```

```
Enter your option2
Forward Display of Workers :
NULL<--[Jim]--><--[Ram]--><--[Tim]--><--[Jon]--><--[Timmy]--><--[Yohan]-->NULL
```

```
Enter your option6

Enter Worker Name : Yohan

Worker Added At end
```

```
Enter your option2
Forward Display of Workers :
NULL<--[Manu]--><--[Jim]--><--[Ram]--><--[Tim]--><--[Jon]--><--[Timmy]--><--[Yohan]-->NULL
```

Singly Circular Linked List

```
 +---------Worker-Menu---------+
 | 1.Create Workers             |
 | 2.Display Workers            |
 | 3.Insert Worker Begining     |
 | 4.Insert Worker Middle       |
 | 5.Insert Worker End          |
 | 6.Delete Worker Begining     |
 | 7.Delete Worker Middle       |
 | 8.Delete Worker End          |
 | 9.Search Worker              |
 | 10.Exit                      |
 +------------------------------+
enter your option
```

```
enter your option1

Enter Name of workerRam
want to add more workers(Y/N)
Enter Name of workerTom
want to add more workers(Y/N)
Enter Name of workerSam
want to add more workers(Y/N)
```

```
enter your option2
[Ram]--->[Tom]--->[Sam]--->(Ram)
```

```
enter your option3

Enter Name of new Worker : Timmy
```

```
enter your option2
[Timmy]--->[Ram]--->[Tom]--->[Sam]--->(Timmy)
```

```
enter your option4
Enter Postion for worker to be inserted : 3

Enter Name of new Worker : Charlie
```

```
enter your option2
[Timmy]--->[Ram]--->[Charlie]--->[Tom]--->[Sam]--->(Timmy)
```

```
enter your option5

Enter Name of new Worker : Max
```

```
enter your option2
[Timmy]--->[Ram]--->[Charlie]--->[Tom]--->[Sam]--->[Max]--->(Timmy)
```

```
enter your option
6

First Worker deleted
```

```
enter your option2
[Ram]--->[Charlie]--->[Tom]--->[Sam]--->[Max]--->(Ram)
```

```
enter your option7
Enter the position of the worker you want to delete: 3

Worker at position 3 deleted
```

```
enter your option
2
[Ram]--->[Charlie]--->[Sam]--->[Max]--->(Ram)
```

```
enter your option8

Last Worker deleted
```

```
enter your option2
[Ram]--->[Charlie]--->[Sam]--->(Ram)
```

```
enter your option9

Enter name of the worker you want to find : Sam

Worker Found at 3
```

Doubly Circular Linked List

```
 +---------Worker-Menu---------+
 | 1.Create Workers            |
 | 2.Display Workers           |
 | 3.Display Workers Reverse   |
 | 4.Insert Worker Begining    |
 | 5.Insert Worker Middle      |
 | 6.Insert Worker End         |
 | 7.Delete Worker Begining    |
 | 8.Delete Worker Middle      |
 | 9.Delete Worker End         |
 | 10.Search Worker            |
 | 11.Exit                     |
 +-----------------------------+
Enter your option
```

```
Enter your option1

Enter Name of worker : Ram
Want to add more Workers(Y/N)
Enter Name of worker : Joy
Want to add more Workers(Y/N)
Enter Name of worker : Manu
Want to add more Workers(Y/N)
```

```
Enter your option2
Forward Display of Workers :
[Ram]--->[Joy]--->[Manu]--->(Ram)
```

```
Enter your option3
Backward Display of Workers :
[Manu]--->[Joy]--->[Ram]--->(Manu)
```

```
Enter your option4

Enter Worker Name John
```

```
Enter your option2
Forward Display of Workers :
[John]--->[Ram]--->[Joy]--->[Manu]--->(John)
```

```
Enter your option5

Enter the position  at which insert Worker3

Enter Worker NameJim
```

```
Enter your option2
Forward Display of Workers :
[John]--->[Ram]--->[Jim]--->[Joy]--->[Manu]--->(John)
```

```
Enter your option6

Enter Worker Name Paul
```

```
Enter your option2
Forward Display of Workers :
[John]--->[Ram]--->[Jim]--->[Joy]--->[Manu]--->[Paul]--->(John)
```

```
Enter your option7
```

```
Enter your option2
Forward Display of Workers :
[Ram]--->[Jim]--->[Joy]--->[Manu]--->[Paul]--->(Ram)
```

```
Enter your option8
Enter the position of the worker you want to delete: 3

Worker at position 3 deleted
```

```
Enter your option2
Forward Display of Workers :
[Ram]--->[Jim]--->[Manu]--->[Paul]--->(Ram)
```

```
Enter your option9
```

```
Enter your option2
Forward Display of Workers :
[Ram]--->[Jim]--->[Manu]--->(Ram)
```

```
Enter your option10
Enter Worker Name to be SearchedJim
Worker found at position : 2
```