**DAY-16(18-11-2024)**

**Data structures**

**SINGLE LINKED LIST:**

* When a member of a structure is a pointer which is pointing to itself is called as self-referencing pointer.

Eg: struct home{

Int value;

Struct home \*ptr;//self referencing pointer

};

Eg1:

struct home1{

Int value;

Struct home2 \*ptr;

// not a self referencing pointer because the home 2 have the different data type

};

struct home2{

float value;

};

Eg2: struct home{

Int value;

Struct home \*ptr;//self referencing pointer

}Home ;

**1)Creating node or home :**

h1

|  |  |
| --- | --- |
| **10** | **\*** |

**100**

**H2**

|  |  |
| --- | --- |
| **20** | **\*** |

**200**

**H3**

|  |  |
| --- | --- |
| **30** | **\*** |

**300**

Home h1;

(//static allocation during compilation ,Home \*h1🡺 this one is dynamic allocation)

Home h2;

Home h3;

**2)Initializing /assigning of nodes:**

h1.value=10;

h2.value=20;

h3.value=30;

h1.ptr=NULL;

h2.ptr=NULL;

h3.ptr=NULL;

A number and a number

Description automatically generated with medium confidence A black and white text with numbers

Description automatically generated with medium confidence

ptr NULL NULL

A number and a number

Description automatically generated with medium confidence

Struct home \*ptr; NULL

Ptr=&h1;

Printf(“%d”,h1.value);🡺10

Printf(“%d”,ptr->value);🡺10

**3)Making relationships /creation of linked list**

h1.ptr=&h2;

h2.ptr=&h3;

**4)Navigating /traversing**

struct home \*ptr1=&h1; 🡺base address of list starting element of the list

printf(“%d”,ptr1->value) 🡺 contentsof(2000)🡺

contents(contents of(100))🡺10

ptr1=&h2; (or) ptr1=h1.ptr;

printf(“%d”,ptr1->value);🡺20

ptr1=&h3; (or) ptr1=h2.ptr;

printf(“%d”,ptr1->value);🡺30

A number with a number on it

Description automatically generated with medium confidenceA white rectangular box with black text

Description automatically generated A white rectangular box with black text

Description automatically generated

NULL

Ptr1

Program :

#include <stdio.h>  
#include <stdlib.h>

typedef struct node  
{  
    int val;  
    struct node \*ptr;  
}NODE;

int main(){  
    NODE h1,h2,h3;  
  
    NODE \*ptr1=NULL;  
  
    h1.val=10;  
    h1.ptr=NULL;  
  
    h2.val=20;  
    h2.ptr=NULL;  
  
    h3.val=30;  
    h3.ptr=NULL;  
  
    printf("\nAddress of h1=%p",&h1);  
    printf("\nAddress of h2=%p",&h2);  
    printf("\nAddress of h3=%p",&h3);  
    printf("\nAddress of ptr1=%p",&ptr1);  
  
    printf("\nh1.value=%d and ptr=%p",h1.val,h1.ptr);  
    printf("\nh2.value=%d and ptr=%p",h2.val,h2.ptr);  
    printf("\nh3.value=%d and ptr=%p",h3.val,h3.ptr);  
    printf("\ncontents of ptr1/ptr is pointing to:%p",ptr1);  
  
    h1.ptr=&h2;  
    h2.ptr=&h3;  
  
    ptr1=&h1;  
   // printf("\n%d,h1.val");  
    printf("\n%d",ptr1->val);  
  
    //ptr1=&h2;  
  
    ptr1=h1.ptr;  
       printf("\n%d",ptr1->val);  
  
    ptr1=h2.ptr;  
    printf("\n%d",ptr1->val);  
  
    printf("\n\n");  
}

**Insertion of node between two nodes:**

Refer notes for diagram

* Before insertion :

h1.ptr=&h2;

h2.ptr=&h3;

* After insertion of h4 between h1 and h2

h1.ptr=&h4;

h4.ptr=&h2;

program:

#include<stdio.h>  
  2 #include<stdlib.h>  
  3  
  4 typedef struct node{  
  5     int val;  
  6     struct node \*ptr;  
  7 }NODE;  
  8  
  9 int main(){  
10     //creating nodes & allocated mem  
11     NODE h1,h2,h3,h4;  
12  
13     //ptr1 is used to traverse between nodes  
14     NODE \*ptr1 = NULL;  
15  
16  
17     //initilaize the values  
18     h1.val = 10;  
19     h1.ptr = NULL;  
20  
21     h2.val = 20;  
22     h2.ptr = NULL;  
23  
24     h3.val = 30;  
25     h3.ptr = NULL;  
26  
27     printf("address of h1 = %p\n",&h1);  
28     printf("address of h2 = %p\n",&h2);  
29     printf("address of h3 = %p\n",&h3);  
30     printf("address of ptr1 = %p\n",&ptr1);  
31  
32     printf("h1.value = %d,h1.ptr = %p\n",h1.val,h1.ptr);  
33     printf("h2.value = %d,h2.ptr = %p\n",h2.val,h2.ptr);  
34     printf("h3.value = %d,h3.ptr = %p\n",h3.val,h3.ptr);  
35     printf("contents of ptr1 is pointing to : %p\n",ptr1);  
36  
37     //making relation  
38     h1.ptr= &h2;  
39     h2.ptr = &h3;  
40  
41     //travesrsing  
42     ptr1= &h1;  
43     printf("%d\n",h1.val);  
44     printf("%d\n",ptr1->val);  
45     ptr1 = h1.ptr;  
46     printf("%d\n",ptr1->val);  
47     ptr1 = h2.ptr;  
48     printf("%d\n",ptr1->val);  
49  
50     //create new node  
51     h4.val = 40;  
52     h4.ptr = NULL;  
53  
54     //inserting new node h4 btw h1 and h2  
55     h1.ptr = &h4;  
56     h4.ptr = &h2;  
57  
58     ptr1=&h1;  
59     printf("%d\n",ptr1->val);  
60     ptr1 = h1.ptr;  
61     printf("%d\n",ptr1->val);  
62     ptr1= h4.ptr;  
63     printf("%d\n",ptr1->val);  
64     ptr1=h2.ptr;  
      printf("%d\n",ptr1->val);

Ptr1=h3.ptr;

If(ptr1==NULL)

{

Printf(“NULL”);

}

Printf(“%d”,ptr1->val);  
      return 0;  
 }

Write program that replace the address(h1,h2,h3) with pointers(ptr1=ptr1->ptr);

Program:

#include <stdio.h>  
#include<stdlib.h>  
typedef struct node  
{  
    int val;  
    struct node \*ptr; //self referencial pointer  
}NODE;  
int main() {  
    //creating nodes & allocated memory  
    //memory is allocated by compiler   
    NODE h1,h2,h3,h4;  
    //ptr1 is used to navigate/traverse between nodes  
    NODE \*ptr1=NULL;  
    //intialize values to each node  
    h1.val=10;  
    h1.ptr=NULL;  
    h2.val=20;  
    h2.ptr=NULL;  
    h3.val=30;  
    h3.ptr=NULL;  
    h4.val=40;  
    h4.ptr=NULL;  
    printf("\nAddress of h1 = %p",&h1);  
    printf("\nAddress of h2 = %p",&h2);  
    printf("\nAddress of h3 = %p",&h3);  
    printf("\nAddress of ptr1 = %p",&ptr1);  
    printf("\nh1.value = %d and ptr= %p",h1.val,h1.ptr);  
    printf("\nh2.value = %d and ptr= %p",h2.val,h2.ptr);  
    printf("\nh3.value = %d and ptr= %p",h3.val,h3.ptr);  
    printf("\nContents of ptr1/ptr1 is pointing to: %p",ptr1);  
    //Making Relationship (traverse)/creating linked list  
    h1.ptr=&h2;  
    h2.ptr=&h3;  
    //traverse or navigate  
    ptr1=&h1;  //ptr1 pointing to base address of list(first record)  
    printf("\n%d",h1.val);  
    printf("\n%d",ptr1->val);  
   // ptr1 = &h2;  
    //ptr1 = h1.ptr;//ptr1 is pointing to h2  
    ptr1=ptr1->ptr;  
    printf("\n%d",ptr1->val);  
    // ptr1 =  &h3;  
     //ptr1 = h2.ptr; //from earlier line  
     ptr1=ptr1->ptr;  
    printf("\n%d",ptr1->val);  
    //inserting new node h4 between h1 and h4  
    h1.ptr=&h4;  
    h4.ptr=&h2;  
    //again ptr1 is pointing to BA of list  
    ptr1=&h1;  
    printf("\nNew list is:\n");  
    printf("\n%d",ptr1->val);  
  
    ptr1=ptr1->ptr;  
    printf("\n%d",ptr1->val);  
  
    ptr1=ptr1->ptr;  
    printf("\n%d",ptr1->val);  
  
    ptr1=ptr1->ptr;  
    printf("\n%d",ptr1->val);  
  
    ptr1=ptr1->ptr;  
    if(ptr1==NULL)  
      printf("\nNULL\n");  
    else  
      printf("\n%d",ptr1->val);  
  
      printf("\nNew list using loops:\n");  
      ptr1=&h1;  
      while(ptr1!=NULL)  
      {  
          printf("%d->",ptr1->val);  
          ptr1=ptr1->ptr;  
      }  
      printf("NULL");  
  
  
  
    printf("\n\n");  
    return 0;  
}

A screenshot of a computer

Description automatically generated

Insertion at the end:

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Output:

A screenshot of a computer

Description automatically generated

Insertion at the beginning refer notes

A computer screen shot of a program

Description automatically generated

A screen shot of a computer program

Description automatically generated

Output:

A screenshot of a computer program

Description automatically generated

**Insertion using functions for printing list:**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

struct node \*ptr; //Self referencial pointer (SELF/SRP)

}NODE;

void printList(NODE \*);

void appendNode(NODE \*,NODE \*);

int main()

{

//creating nodes & allocated mem

NODE h1,h2,h3,h4;

//ptr1 is used to navigate/travese betwn nodes

NODE \*ptr1=NULL;

//initialze values to each node

h1.val = 10;

h1.ptr = NULL;

h2.val = 20;

h2.ptr = NULL;

h3.val = 30;

h3.ptr = NULL;

//Making relationship (travese)/creating LList

h1.ptr = &h2;

h2.ptr = &h3;

//traversing the list

ptr1 = &h1; // ptr1 is pointing to BA of list

printList(ptr1); //Send the BA of list to function

h4.val = 40;

h4.ptr = NULL;

//h3.ptr = &h4;

ptr1 = &h1;

//traversing the list

ptr1 = &h1; // ptr1 is pointing to BA of list

printList(ptr1); //Send the BA of list to function

printf("\n\n");

return 0;

}

void printList(NODE \*head)

{

printf("\nlist using loops and function\n");

while(head!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

printf("NULL");

}

**Using functions for inserting (just appending in the present list):**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

struct node \*ptr; //Self referencial pointer (SELF/SRP)

}NODE;

void printList(NODE \*);

void appendNode(NODE \*,NODE \*);

int main()

{

//creating nodes & allocated mem

NODE h1,h2,h3,h4;

//ptr1 is used to navigate/travese betwn nodes

NODE \*ptr1=NULL;

//initialze values to each node

h1.val = 10;

h1.ptr = NULL;

h2.val = 20;

h2.ptr = NULL;

h3.val = 30;

h3.ptr = NULL;

//Making relationship (travese)/creating LList

h1.ptr = &h2;

h2.ptr = &h3;

//traversing the list

ptr1 = &h1; // ptr1 is pointing to BA of list

printList(ptr1); //Send the BA of list to function

h4.val = 40;

h4.ptr = NULL;

//h3.ptr = &h4;

ptr1 = &h1;

appendNode(ptr1,&h4);

//traversing the list

ptr1 = &h1; // ptr1 is pointing to BA of list

printList(ptr1); //Send the BA of list to function

printf("\n\n");

return 0;

}

void printList(NODE \*head)

{

printf("\nlist using loops and function\n");

while(head!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

printf("NULL");

}

void appendNode(NODE \*head,NODE \*nn)

{

printf("\nIn appendNode\n");

while(head->ptr!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

printf("\n%d",head->val);

printf("\n%p",head->ptr);

head->ptr = nn; //adding node to the end of the list

}

**Using functions inserting the node in an empty list:**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

struct node \*ptr; //Self referencial pointer (SELF/SRP)

}NODE;

void printList(NODE \*);

NODE\* appendNode(NODE \*,NODE \*);

int main()

{

//creating nodes & allocated mem

NODE h1,h2,h3,h4;

//ptr1 is used to navigate/travese betwn nodes

NODE \*head=NULL;

h4.val = 40;

h4.ptr = NULL;

head = appendNode(head,&h4);

//traversing the list

printList(head);

h1.val = 10;

h1.ptr = NULL;

h2.val = 20;

h2.ptr = NULL;

h3.val = 30;

h3.ptr = NULL;

head = appendNode(head,&h1);

appendNode(head,&h2);

appendNode(head,&h3);

//traversing the list

printList(head);

printf("\n\n");

return 0;

}

void printList(NODE \*head)

{

printf("\nlist using loops and function\n");

while(head!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

printf("NULL");

}

NODE \*appendNode(NODE \*head,NODE \*nn)

{

printf("\nIn appendNode\n");

NODE \*temp = head;

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

printf("\nList is not empty\n");

while(head->ptr!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

head->ptr = nn; //adding node to the end of the list

}

head = temp; //head is again pointed to the BA

return head;

}

**Insertion of the node at the beginning of the list(in a empty list):**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

struct node \*ptr; //Self referencial pointer (SELF/SRP)

}NODE;

void printList(NODE \*);

NODE\* appendNode(NODE \*,NODE \*);

NODE \*addNodeBeg(NODE \*, NODE \*);

int main()

{

//creating nodes & allocated mem

NODE h1,h2,h3,h4;

//ptr1 is used to navigate/travese betwn nodes

NODE \*head=NULL;

h4.val = 40;

h4.ptr = NULL;

//head = appendNode(head,&h4);

//traversing the list

printList(head);

h1.val = 10;

h1.ptr = NULL;

h2.val = 20;

h2.ptr = NULL;

h3.val = 30;

h3.ptr = NULL;

/\*head = appendNode(head,&h1);

appendNode(head,&h2);

appendNode(head,&h3);

\*/

head = addNodeBeg(head,&h1);

//traversing the list

printList(head);

head = addNodeBeg(head,&h2);

//traversing the list

printList(head);

printf("\n\n");

return 0;

}

void printList(NODE \*head)

{

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

return;

}

printf("\nlist using loops and function\n");

while(head!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

printf("NULL");

}

NODE \*appendNode(NODE \*head,NODE \*nn)

{

printf("\nIn appendNode\n");

NODE \*temp = head;

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

printf("\nList is not empty\n");

while(head->ptr!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

head->ptr = nn; //adding node to the end of the list

}

head = temp; //head is again pointed to the BA

return head;

}

NODE \*addNodeBeg(NODE \*head, NODE \*nn)

{

NODE \*temp = head;

printf("\nIn addNodeBeg\n");

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

nn->ptr = head;

head = nn; // making the head point to the nn(BA of new list)

}

return head; //return the BA of the list

}

**Deletion of Node:**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

struct node \*ptr; //Self referencial pointer (SELF/SRP)

}NODE;

void printList(NODE \*);

NODE\* appendNode(NODE \*,NODE \*);

NODE \*addNodeBeg(NODE \*, NODE \*);

int delNode(NODE \*, int);

int main()

{

//creating nodes & allocated mem

NODE h1,h2,h3,h4;

//ptr1 is used to navigate/travese betwn nodes

NODE \*head=NULL;

h4.val = 40;

h4.ptr = NULL;

//head = appendNode(head,&h4);

//traversing the list

printList(head);

h1.val = 10;

h1.ptr = NULL;

h2.val = 20;

h2.ptr = NULL;

h3.val = 30;

h3.ptr = NULL;

/\*head = appendNode(head,&h1);

appendNode(head,&h2);

appendNode(head,&h3);

\*/

head = addNodeBeg(head,&h4);

//traversing the list

printList(head);

head = addNodeBeg(head,&h3);

//traversing the list

printList(head);

head = addNodeBeg(head,&h2);

//traversing the list

printList(head);

head = addNodeBeg(head,&h1);

//traversing the list

printList(head);

//head = &h4;

if(delNode(head,30) == 0)

printf("\nNode Deleted\n");

else

printf("\nKey Not found in the list\n");

//traversing the list

printList(head);

printf("\n\n");

return 0;

}

void printList(NODE \*head)

{

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

return;

}

printf("\nlist using loops and function\n");

while(head!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

printf("NULL");

}

NODE \*appendNode(NODE \*head,NODE \*nn)

{

printf("\nIn appendNode\n");

NODE \*temp = head;

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

printf("\nList is not empty\n");

while(head->ptr!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

head->ptr = nn; //adding node to the end of the list

}

head = temp; //head is again pointed to the BA

return head;

}

NODE \*addNodeBeg(NODE \*head, NODE \*nn)

{

NODE \*temp = head;

printf("\nIn addNodeBeg\n");

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

nn->ptr = head;

head = nn; // making the head point to the nn(BA of new list)

}

return head; //return the BA of the list

}

int delNode(NODE \*head, int key)

{

int flag = 1;

NODE \*temp = head;

while(head!=NULL)

{

if(head->val == key)

{

//found

flag = 0;

break;

}

temp = head;

head = head->ptr;

}

if(flag == 0)

{

printf("\nGunashree->%d",temp->val);

printf("\nPooja->%d",head->val);

temp->ptr = head->ptr;

}

return flag;

}

Output: list using loops and function

40->NULL

In addNodeBeg

list using loops and function

30->40->NULL

In addNodeBeg

list using loops and function

20->30->40->NULL

In addNodeBeg

list using loops and function

10->20->30->40->NULL

Gunashree->20

Pooja->30

Node Deleted

list using loops and function

10->20->40->NULL

**Deletion simple code:**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

struct node \*ptr; //Self referencial pointer (SELF/SRP)

}NODE;

void printList(NODE \*);

int deleteNode(NODE \*,int);

int main()

{

//creating nodes & allocated mem

NODE h1,h2,h3,h4;

//ptr1 is used to navigate/travese betwn nodes

NODE \*ptr1=NULL;

//initialze values to each node

h1.val = 10;

h1.ptr = NULL;

h2.val = 20;

h2.ptr = NULL;

h3.val = 30;

h3.ptr = NULL;

//Making relationship (travese)/creating LList

h1.ptr = &h2;

h2.ptr = &h3;

//traversing the list

ptr1 = &h1; // ptr1 is pointing to BA of list

printList(ptr1); //Send the BA of list to function

h4.val = 40;

h4.ptr = NULL;

//h3.ptr = &h4;

ptr1 = &h1;

//traversing the list

ptr1 = &h1; // ptr1 is pointing to BA of list

printList(ptr1); //Send the BA of list to function

deleteNode(ptr1,30);

printList(ptr1);

printf("\n\n");

return 0;

}

void printList(NODE \*head)

{

printf("\nlist using loops and function\n");

if(head==NULL)

{

printf("empty");

}

while(head!= NULL)

{

printf("%d->",head->val);

head = head->ptr;

}

printf("NULL");

}

int deleteNode(NODE \*head,int key)

{

int flag=1;

NODE \*temp=head;

while(head!=NULL)

{

if(head->val==key)

{

flag=0;

break;

}

temp=head;

head=head->ptr;

}

if(flag==0)

{

temp->ptr=head->ptr;

}

return flag;

}

**Optimize of linked list(using malloc or dynamic allocation) :**

**Program:**

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Question: 1)delete a node at starting and delete a node at end 2)we need to free() each node (from GitHub v07.c) 3) memory allocation

Program (change this program according to the above question):

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

struct node \*next; //Self referencial pointer (SELF/SRP)

}NODE;

NODE \*createNode(NODE \*);

int dispMainMenu();

void printList(NODE \*);

NODE\* appendNode(NODE \*,NODE \*);

NODE \*addNodeBeg(NODE \*, NODE \*);

int delNode(NODE \*, int);

int main()

{

NODE \*nn=NULL;

NODE \*head=NULL,\*temp=NULL;

int flag=0;

int key=0;

while(1)

{

switch(dispMainMenu())

{

case 1:

nn = createNode(nn);

head = addNodeBeg(head,nn);

break;

case 2:

nn = createNode(nn);

head = appendNode(head,nn);

break;

case 3:

printf("\nEnter the key of the node: ");

scanf("%d",&key);

delNode(head,key);

break;

case 4:

printList(head);

break;

case 5:

flag = 1;

break;

default:

printf("\nEnter the correct Choice");

break;

}

if(flag == 1)

break;

}

printf("\n\n");

return 0;

}

NODE \*createNode(NODE \*nn)

{

nn = (NODE \*)malloc(sizeof(NODE));

printf("\nEnter the value of New Node: ");

scanf("%d",&nn->val);

nn->next = NULL;

return nn;

}

int dispMainMenu()

{

int ch;

printf("\nLinked List Menu\n");

printf("\nPress,");

printf("\n1. Add Node in Begning");

printf("\n2. Add Node to the End");

printf("\n3. Delete a Node");

printf("\n4. Print List");

printf("\n5. Exit");

printf("\nChoice: ");

scanf("%d",&ch);

return ch;

}

void printList(NODE \*head)

{

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

return;

}

printf("\nList is\n===================================\n");

while(head!= NULL)

{

printf("%d->",head->val);

head = head->next;

}

printf("NULL");

printf("\n\n===================================\n");

}

NODE \*appendNode(NODE \*head,NODE \*nn)

{

// printf("\nIn appendNode\n");

NODE \*temp = head;

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

// printf("\nList is not empty\n");

while(head->next!= NULL)

{

// printf("%d->",head->val);

head = head->next;

}

head->next = nn; //adding node to the end of the list

}

head = temp; //head is again pointed to the BA

return head;

}

NODE \*addNodeBeg(NODE \*head, NODE \*nn)

{

NODE \*temp = head;

// printf("\nIn addNodeBeg\n");

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

nn->next = head;

head = nn; // making the head point to the nn(BA of new list)

}

return head; //return the BA of the list

}

int delNode(NODE \*head, int key)

{

int flag = 1;

NODE \*temp = head;

while(head!=NULL)

{

if(head->val == key)

{

//found

flag = 0;

break;

}

temp = head;

head = head->next;

}

if(flag == 0)

{

temp->next = head->next;

}

return flag;

}

DOUBLY LINKED LIST:

* Refer notes for diagram
* In all the codes head is ptr1 and next is ptr
* Append at the end

head->next = nn; //adding node to the end of the list

nn->prev = head;

* For backward traversing (head=head->prev)
* Question: Delete a nodes

Program for doubly linked list:

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

struct node \*next; //Self referencial pointer (SELF/SRP)

struct node \*prev;

}NODE;

NODE \*createNode(NODE \*);

int dispMainMenu();

void printListFD(NODE \*);

void printListBD(NODE \*);

NODE\* appendNode(NODE \*,NODE \*);

NODE \*addNodeBeg(NODE \*, NODE \*);

int delNode(NODE \*, int);

int main()

{

NODE \*nn=NULL;

NODE \*head=NULL,\*temp=NULL;

int flag=0;

int key=0;

while(1)

{

switch(dispMainMenu())

{

case 1:

nn = createNode(nn);

head = addNodeBeg(head,nn);

break;

case 2:

nn = createNode(nn);

head = appendNode(head,nn);

break;

case 3:

printf("\nEnter the key of the node: ");

scanf("%d",&key);

delNode(head,key);

break;

case 4:

printListFD(head);

break;

case 5:

printListBD(head);

break;

case 6:

flag = 1;

break;

default:

printf("\nEnter the correct Choice");

break;

}

if(flag == 1)

break;

}

printf("\n\n");

return 0;

}

NODE \*createNode(NODE \*nn)

{

nn = (NODE \*)malloc(sizeof(NODE));

printf("\nEnter the value of New Node: ");

scanf("%d",&nn->val);

nn->next = NULL;

nn->prev = NULL;

return nn;

}

int dispMainMenu()

{

int ch;

printf("\nLinked List Menu\n");

printf("\nPress,");

printf("\n1. Add Node in Begning");

printf("\n2. Add Node to the End");

printf("\n3. Delete a Node");

printf("\n4. Print List FD");

printf("\n5. Print List BD");

printf("\n6. Exit");

printf("\nChoice: ");

scanf("%d",&ch);

return ch;

}

void printListFD(NODE \*head)

{

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

return;

}

printf("\nList is\n===================================\n");

while(head!= NULL)

{

printf("%d->",head->val);

head = head->next;

}

printf("NULL");

printf("\n\n===================================\n");

}

void printListBD(NODE \*head)

{

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

return;

}

printf("\nList is\n===================================\n");

//travese until last node

while(head->next!= NULL)

{

head = head->next;

}

//print from the last to first node

do

{

printf("%d->",head->val);

head = head->prev;

}while(head!=NULL);

printf("NULL");

printf("\n\n===================================\n");

}

NODE \*appendNode(NODE \*head,NODE \*nn)

{

// printf("\nIn appendNode\n");

NODE \*temp = head;

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

// printf("\nList is not empty\n");

while(head->next!= NULL)

{

// printf("%d->",head->val);

head = head->next;

}

head->next = nn; //adding node to the end of the list

nn->prev = head;

}

head = temp; //head is again pointed to the BA

return head;

}

NODE \*addNodeBeg(NODE \*head, NODE \*nn)

{

NODE \*temp = head;

// printf("\nIn addNodeBeg\n");

if(head == NULL)

{

//the list is empty

printf("\nList is Empty\n");

head = nn;

temp = nn;

}

else

{

nn->next = head;

head->prev = nn;

head = nn; // making the head point to the nn(BA of new list)

}

return head; //return the BA of the list

}

int delNode(NODE \*head, int key)

{

int flag = 1;

NODE \*temp = head;

while(head!=NULL)

{

if(head->val == key)

{

//found

flag = 0;

break;

}

temp = head;

head = head->next;

}

if(flag == 0)

{

temp->next = head->next;

}

return flag;

deletion at beginning in doubly linked list:

#include <stdio.h>  
#include <stdlib.h>

typedef struct node  
{  
  int val;  
  struct node \*next; //Self referencial pointer (SELF/SRP)  
  struct node \*prev;  
}NODE;

NODE \*createNode(NODE \*);  
void printListFD(NODE \*);  
NODE\* appendNode(NODE \*,NODE \*);  
NODE \*addNodeBeg(NODE \*, NODE \*);  
int delNode(NODE \*, int);  
NODE \*delNodeBeg(NODE \*);

int main(){  
    NODE \*nn=NULL;  
    NODE \*head = NULL;  
    NODE \*temp = NULL;  
  
    nn=createNode(nn);  
    head = addNodeBeg(head,nn);  
    nn=createNode(nn);  
    head = addNodeBeg(head,nn);  
    nn=createNode(nn);  
    head = addNodeBeg(head,nn);  
    printListFD(head);  
    head = delNodeBeg(head);  
    printListFD(head);  
}

NODE \*createNode(NODE \*nn)  
{  
  nn = (NODE \*)malloc(sizeof(NODE));  
  printf("\nEnter the value of New Node: ");  
  scanf("%d",&nn->val);  
  nn->next = NULL;  
  nn->prev = NULL;  
  return nn;  
}

void printListFD(NODE \*head)  
{

  if(head == NULL)  
  {  
    //the list is empty  
    printf("\nList is Empty\n");  
    return;  
  }  
  printf("\nList is\n===================================\n");  
  while(head!= NULL)  
  {  
    printf("%d->",head->val);  
    head = head->next;    
  }  
  printf("NULL");  
  printf("\n\n===================================\n");  
}

NODE \*appendNode(NODE \*head,NODE \*nn)  
{  
  // printf("\nIn appendNode\n");  
  NODE \*temp = head;

  if(head == NULL)  
  {  
    //the list is empty  
    printf("\nList is Empty\n");  
    head = nn;  
    temp = nn;  
  }  
  else  
  {  
    // printf("\nList is not empty\n");  
    while(head->next!= NULL)  
    {  
      // printf("%d->",head->val);  
      head = head->next;    
    }  
  
    head->next = nn; //adding node to the end of the list  
    nn->prev = head;  
  }  
  head = temp; //head is again pointed to the BA

  return head;  
}

NODE \*addNodeBeg(NODE \*head, NODE \*nn)  
{  
  NODE \*temp = head;  
  // printf("\nIn addNodeBeg\n");  
  if(head == NULL)  
  {  
    //the list is empty  
    printf("\nList is Empty\n");  
    head = nn;  
    temp = nn;  
  }  
  else  
  {  
    nn->next = head;  
    head->prev = nn;  
    head = nn;     // making the head point to the nn(BA of new list)  
  }  
  return head; //return the BA of the list  
}

int delNode(NODE \*head, int key)  
{  
  int flag = 1;

  NODE \*temp = head;

  while(head!=NULL)  
  {  
    if(head->val == key)  
    {  
      //found  
      flag = 0;  
      break;  
    }  
    temp = head;  
    head = head->next;  
  }

  if(flag == 0)  
  {  
  
    temp->next = head->next;  
  }

  return flag;  
}

NODE \*delNodeBeg(NODE \*head){  
    NODE \*temp = head;  
    head = head->next;  
    if(head != NULL)  
    head->prev = NULL;  
    free(temp);  
    return head;  
}

}

CIRCULAR LINKED LIST:

* Refer notes
* nn->next=temp;

Temp->prev=nn;

* Question: insertion and deletion

Question in employee project do the menu based in employee