**DATA STRUCTURES**

When a pointer points to itself it is called as Self referential pointer.

Only for navigation we use Self referential pointer.

**LINKED LIST:**

1.Creating Homes

2.Intializing the values

3.Making Relationships

**EX:**

#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;

//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;

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

printf("\n%d",ptr1->val);

// ptr1 = &h3;

ptr1 = h2.ptr; //from earlier line

printf("\n%d",ptr1->val);

printf("\n\n");

return 0;

}

**EX:**

#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;

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

printf("\n%d",ptr1->val);

// ptr1 = &h3;

ptr1 = h2.ptr; //from earlier line

printf("\n%d",ptr1->val);

h4.val=40;

h4.ptr=NULL;

//inserting new node h4 between h1 and h2

h1.ptr=&h4;

h4.ptr=&h2;

//again ptr1 is pointing to base address of list

ptr1= &h1;

printf("\nNew list is\n");

printf("\n%d->",ptr1->val);

ptr1=h1.ptr;

printf("%d->",ptr1->val);

ptr1=h4.ptr;

printf("%d->",ptr1->val);

ptr1=h2.ptr;

printf("%d->",ptr1->val);

ptr1=h3.ptr;

if(ptr1==NULL)

printf("NULL");

else

printf("\n%d",ptr1->val);

printf("\n\n");

return 0;

}

**EX:**

#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;

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

printf("\nptr1=>h2\n");

ptr1=ptr1->ptr;

printf("\n%d",ptr1->val);

// ptr1 = &h3;

// ptr1 = h2.ptr; //from earlier line

ptr1=ptr1->ptr;

printf("\n%d",ptr1->val);

h4.val=40;

h4.ptr=NULL;

//inserting new node h4 between h1 and h2

h1.ptr=&h4;

h4.ptr=&h2;

//again ptr1 is pointing to base address of list

ptr1= &h1;

printf("\nNew list is\n");

printf("\n%d->",ptr1->val);

ptr1=ptr1->ptr;

printf("%d->",ptr1->val);

ptr1=ptr1->ptr;

printf("%d->",ptr1->val);

ptr1=ptr1->ptr;

printf("%d->",ptr1->val);

ptr1=ptr1->ptr;

if(ptr1==NULL)

printf("NULL");

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;

}

**EX:**

h5.val=50;

h5.ptr=NULL;

//inserting new node at last

h3.ptr=&h5;

ptr1= &h1;

printf("\nNew list is\n");

printf("\n%d->",ptr1->val);

ptr1=ptr1->ptr;

printf("%d->",ptr1->val);

ptr1=ptr1->ptr;

printf("%d->",ptr1->val);

ptr1=ptr1->ptr;

printf("%d->",ptr1->val);

ptr1=ptr1->ptr;

printf("%d->",ptr1->val);

ptr1=ptr1->ptr;

if(ptr1==NULL)

printf("NULL");

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;

}

**EX: BY USING FUNCTIONS**

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

{

printf("\nIn appendNode");

while(head->ptr!=NULL)

{

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

head=head->ptr;

}

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

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

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

}

**EX: DELETING A 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;

}

**EX: APPENDING THE CODE USING DO WHILE**

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

int val;

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

}NODE;

void printList(NODE \*);

void printListAdd(NODE \*);

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

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

int delNode(NODE \*, int);

int main()

{

NODE \*nn=NULL;

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

int ch;

do

{

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

printf("\nNew Node Add=%p",nn);

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

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

nn->next = NULL;

head = appendNode(head,nn);

if (head == NULL)

{

printf("\nUnable to create Node\n");

printf("\nMem allocation failed\n");

return 1;

}

printf("\nDo you want to add new node(1/0): ");

scanf("%d",&ch);

}while(ch !=0);

printListAdd(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->next;

}

printf("NULL");

}

void printListAdd(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("\n%d->%p",head->val,head->next);

head = head->next;

}

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->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;

}

**EX:**

Freeing the contents of the node.

Valgrind