**Cursor Based Implementation**

* Some programming languages doesn’t support pointers.
* But we need linked list representation to store data.
* Solution:
  + **Cursor implementation** – implementing linked list on an array.
* Main idea is:
  + Convert a linked list based implementation to an array based implementation.
  + Instead of pointers, array index could be used to keep track of node.
  + Convert the node structures (collection of data and next pointer) to a global array of structures.

Need to find a way to perform dynamic memory allocation performed using malloc and free functions.

Operations:

1.Initialization of Array

2.Insertion:Insert new element at position pointed by header(ZERO) and assign new position which is null to header

3.Deletion:Delete an element and assign that position to header(ZERO)

4.Traversal:Display the entire Array

Program:

#include<stdio.h>  
#include<conio.h>  
#define SIZE 11  
struct Node  
{  
   char element;  
   int nextpos;  
};  
typedef struct Node cursor;

cursor cursorspace[SIZE];

void main()  
{  
  
int choice,pos;

char ch;  
clrscr();

initialize();

do

{

printf("\n1. Insert \n2. Delete \n3. Display \n4. Exit ");  
printf("\nEnter ur choice:\t");  
scanf("%d",&choice);  
switch(choice)  
{  
case 1:  
 printf(“Enter any Character”);

scanf(“%d”,&ch);

insert(ch);

break;  
case 2:  
 printf(“Enter any Character to delete”);

scanf(“%d”,&ch);

del (ch);

break;  
case 3:  
   display();

    break;  
case 4:

break;

}

}while(choice!=4);

getch();

}

void initialize()  
{  
   int i;  
   for(i=0;i<=SIZE-1;i++)  
   {  
      cursorspace[i].nextpos=i+1;  
      cursorspace[i].element=x;  
   }

cursorspace[i].nextpos=0;  
      cursorspace[i].element=x;  
}

void Insert(char X)  
{  
      int Temp;  
      Temp=CursorAlloc();  
      if(Temp==-1)  
      printf("\nOut of space");  
      else

cursorspace[temp].element==x);

 }  
int CursorAlloc()  
{  
   int P;  
   P=cursorspace[0].nextpos;  
   cursorspace[0].nextpos=cursorspace[P].nextpos;  
   return P;  
}

Void del(char x)

{

Int p,tmp;

P=findprevious(x);

If(p==0)

{

tmp=1;

Cursorfree(tmp);

}

else

{

tmp=cursorspace[p].nextpos;

cursorspace[p].nextpos=cursorspace[tmp.nextpos;

cursorfree(tmp);

}

}

int findprevious(char x)

{

int p=0;

if(cursorspace[1].element==x)

{

return 0;

}

for(int i=0;x!=cursorspace[i].element;i++)

{

p=cursorspace[i].nextpos;

}

for(i=0;cursorspace[i].nextpos!=p;i++)

{

p=I;

return p;

}

void cursorfree(int p)

{

for(int i=cursorspace[p.nextpos;i<SIZE-1;i++)

{

If(cursorspace[i].nextpos==cursorspace[0].nextpos)

{

cursorspace[i].nextpos=p;

break;

}

}

cursorspace[p].nextpos=cursorspace[0].nextpos;

cursorspace[p].element=’x’;

cursorspace[0].nextpos=p;

}

void Delete(int X,LIST L)  
{  
   int P,Temp;  
   P=FindPrevious(X,L);  
   if(!IsLast(P))  
   {  
      Temp=cursorspace[P].nextpos;  
      cursorspace[P].nextpos=cursorspace[Temp].nextpos;  
      CursorFree(Temp);  
   }  
}