DSA
INTERNSHALA
Linear Linked List

Basics of Linear Linked List

There are three types of linked list

- Linear / sequential linked list
- Circular linked list
- Doubly linked list

Basic components of linked list

- Head/Start a pointer to the first node
- Data and link/next attributes for each node
- For doubly linked node one more attribute prev is used along with next
- In last node link/next attribute will contain null as its value.
- In Circular linked node last node contains address of first node in next

[Algorithm to traverse the nodes of a linked list]

1. Set temp:=start
2. Repeat steps 3,4 while temp != NULL
3. Write:=data[temp]
4. Set temp:=link[temp]
   [end of while loop]
5. return

[Algorithm to insert a new node in the beginning of a linear linked list]

Assumption: free memory comprises of an empty linked list

1. if avail=NULL , then: [avail is a linked list of empty nodes]
   write: overflow error
   return
   [end of if]
2. set new:=avail [pick up the address of the first node in the available linked list]
3. set avail:=link[avail] [now avail will point to the empty node which was previously second]
4. set data[new]:=val [assign the value to new node]
5. set link[new]:=start [assign value of start that is address of first node which will now be second node]
6. set start:=new [updating value of start]
7. return

[we donot deal with memory allocation in algorithms]

[Algorithm to insert new node before a given node in linear linked list]

1. if avail=NULL, then:
   write:=overflow error
   exit
2. set new:=avail [pick up the address of the first node in the available linked list]
3. set avail:=link[avail] [now avail will point to the empty node which was previously second]
4. set data[new] = val [assign the value to new node]
5. if item=data[start], them: [if new node is to be inserted in the beginning]
   call insertAtBeggining
   exit
6. set temp:=link[start], prev:=start
7. repeat steps 8,9 while temp != NULL
8. if item=data[temp], then:
   set link[prev]:=new
   set link[new]:=temp
   exit
   [end of if]
9. set prev:=temp, temp:=link[temp]
   [end of while]
10. if temp=NULL, then:
   write: No such node with value item found

[end of if]
11. exit

[insertAtBeggining]

1. set link[new]:=start [assign value of start that is address of first node which will be now second node]
2. set start:=new [updating value of start]
3. return

[Algorithm to insert a new node after a given node in a linear linked list]

1. if avail=NULL, then:
   write: overflow error
   return
   [end of if]
2. set new:=avail
3. set avail:=link[avail]
4. set data[new]:=val
5. set temp:=start
6. repeat steps 7,8 while temp != NULL
7. if item=data[temp], then:
   set link[new]:=link[temp]
   set link[temp]:=new
   return
   [end of if]
8. temp:= link[temp]
9. if temp=NULL, then
   write: no such node with value item found
   [end of if]
10. return

[Algorithm to delete the first node]

1. set temp:=start [preserve the address of the first node into temp]
2. set start:=link[start] [now update the value of start]
3. set link[temp]:=avail [assigning the address of first empty node of available linked list to link of removed node]

4. set avail:=temp [update the value of temp]
5. return

[Algorithm to delete a given node]

1. if item=data[start], then:
   call deletefirst
   exit
   [end of if]
2. set temp:=link[start], prev:=start
3. repeat steps 4,5 while temp != NULL
4. if item=data[temp], then:
   set link[prev]:=link[temp]
   set link[temp]:=avail
   set avail:=temp
   exit
   [end of if]
5. set prev:=temp, temp:=link[temp]
6. write: no such value exists
7. exit

[deletefirst]

1. set temp:=start
2. set start:=link[start]
3. set link[temp]:=avail
4. set avail:=temp
5. return

Creating and printing of a linear linked list

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

typedef struct linklist {
 int data;
 struct linklist *link;
} node;

char ch;
```

```
node *create(node *start) {
 node *temp, *p;
 ch='y';

 printf("\n\t\t *** INPUT BLOCK ***\n");
 if(start != NULL) {
  printf("\n\n\t **** LIST ALREADY EXIST ****\n");
  getch();
  return(start);
 }
 fflush(stdin);
 while(ch == 'y') {
  temp=(node *) malloc(sizeof(node));
  printf("\n\t Enter the no :➔ ");
  scanf("%d",&temp->data);
  temp->link = NULL;
  if(start == NULL) {
   p = start = temp;
  } else {
   p->link = temp;
   p=temp;
  }
  fflush(stdin);
  printf("\t Do you want to continue (y/n) ? :");
  ch = getchar();
 }
 return(start);
}

void print(node *start){
 node *temp;
 system("cls");
 printf("\n\n Base address Number link");
 printf("\n============================");
 for(temp=start; temp != NULL; temp=temp->link) {
  printf("\n%10u %10d %10u",temp,temp->data,temp->link);
 }
 printf("\n\n\t Press any key to goto MAIN BLOCK......");
 getch();
 return;
}

int main() {
 node *start;

 start=NULL:
 start = create(start);
 print(start);
}
```

Inserting a new node in the beginning of a linear linked list

```
node *insf(int item,node *start) {
 node *p;
 p = (node *) malloc(sizeof(node));
 p->data=item;
 p->link=start;
 start=p;
 printf("\n\n\t Element is successfully inserted");
 getch();
 return(start);
}
```

Searching a particular node in a linear linked list

```
node *search(int item, node *start) {
 node *temp;
 for(temp = start; temp != NULL; temp = temp->link) {
  if(temp->data == item)
   return(temp);
 }
 return(NULL);
}
```

Inserting a new node before a given node

```
node *insb(int item, int var, node *start) {
 node *temp, *p1, *p;
 p = (node *) malloc(sizeof(node));
 for(temp=start;(temp->data != var) && (temp != NULL);temp= temp->link)
{
  p1=temp;
 }
 if(temp == NULL) {
  printf("\n\n\t Number not found in the Linklist");
  getch();
 } else if(temp == start) {
  start = insf(item,p,start);
 } else {
  p->data = item;
  p->link = p1->link;
  p1->link = p;
  printf("\n\n\t Element is successfully inserted");
  getch();
 }
 return(start);
}
```

Inserting a node after a given node

```
void insa(int item, int var, node *start) {
 node *temp, *p;
 p = (node *) malloc(sizeof(node));
 for(temp=start; (temp->data != var)&&(temp != NULL) {
  temp=temp->link;
 }
 if(temp == NULL) {
  printf("\n\n\t Number not found in the linkedlist");
  getch();
  return;
 } else {
  p->data = item;
  p->link=temp->link;
  temp->link=p;
  printf("\n\n\t Element is successfully inserted");
  getch();
 }
 return;
}
```

Inserting a new node at the end

```
void inse(int item, node *start) {
 node *temp, *p;
 p = (node *) malloc(sizeof(node));
 p->data = item;

 for(temp=start; temp->link != NULL; temp=temp->link);

 temp->link=p;
 p->link=NULL;
 printf("\n\n\t Element is successfully inserted");
 getch();
 return;
}
```

Deleting the first node

```
node *delf(node *start) {
 node *tmp = start;
 start = start->link;
 printf("\n\n\t Element is successfully deleted");
 getch();
 free(temp);
 return(start);
}
```

Deleting a particular node

```
node *delp(int item, node *start) {
 node *temp , *temp1;

 for(temp = start; (temp->data != item) && (temp != NULL); temp=temp-
>link) {
  temp1=temp;
 }

 if(temp == NULL) {
  printf("\n\n\t Number not found in the Linkedlist");
  getch();
  return(start);
 }
 if(temp == start) {
  start = delf(start);
 } else {
  temp1->link = temp1->link->link;
  printf("\n\n\t Element is successfully deleted");
  getch();
  free(temp);
 }
 return(start);
}
```

Deleting the last node

```
void dele(node *start) {
 node *temp, *temp1;
 for(temp=start; temp->link != NULL; temp=temp->link) {
  temp1=temp;
 }

 temp1->link=NULL;
 free(temp);
 printf("\n\n\t Element is successfully deleted");
 getch();
 return;
}
```