Routines for Single, Double and Circular Linked List

Operations	Single Linked List	Doubly Linked List	Circular Linked List
Structure	Generally data in the single linked list are maintained in the form of node in which each node consists of data and link Link represents address to the next node.	• Generally data in the doubly linked list are maintained in the form of node in which each node consists of data and forward link and backward link • Forward Link represents address to the next node. And backward link represents the address of previous node in the list	Generally data in the circular linked list are maintained in the form of node in which each node consists of data and link Link represents address to the next node. The last node link of the circular linked list points to the address of the first node.

	struct node
	{
	int data;
	struct node *link;
	}
		struct dnode
		{
		int data;
		struct dnode *flink,*blink;
		}
			struct node
			{
			int data;
			struct node *link;
			}

Find	• This function is used to identify the position for the corresponding data in the list • This function is used by the insert and delete operation in the list • In which while inserting the data into the list, it need the location(Address) to store the after the location. • It accepts data and the head address as parameters.		

	struct node *find(int ele, struct node *head)
	{
	struct node *p;
	p=head➔link;
	while(p!=NULL)
	{
	if(p➔data==ele)
	return p;
	else
	p=p➔link;
	}
	return NULL;
	}
		struct dnode *find(int ele, struct node *head)
		{
		struct dnode *p,*q;
		p=head➔flink;
		while(p!=NULL)
		{
		if(p➔data==ele)
		return p;
		else
		p=p➔flink;
		}
		return NULL;
		}
			struct node *find(int ele, struct node *head)
			{
			struct node *p;
			p=head➔link;
			while(p!=NULL)
			{
			if(p➔data==ele)
			return p;
			else
			p=p➔link;
			}
			return NULL;
			}

Operations	**Single Linked List**	**Doubly Linked List**	**Circular Linked List**
Find previous	• This function is used to identify the previous node address (position) for the corresponding node. • This information is used to perform delete operation so as to maintaining the list we are in need of previous node to be linked with next node for the deleted node. • This function is not for doubly linked list since each consists of blink and flink.		

	struct node *findpre(int ele,struct node *head)
	{
	struct node *pre,*p;
	pre=head;
	p=head➔link;
	while(p!=null)
	{
	if(p➔data==x)
	return pre;
	else
	{
	pre=p;
	p=p➔link;
	}
	}
	return NULL;
	}
		struct dnode *findpre(int ele,struct node *head)
		{
		struct dnode *pre,*p;
		pre=head;
		p=head➔flink;
		while(p!=null)
		{
		if(p➔data==x)
		return pre;
		else
		{
		pre=p;
		p=p➔flink;
		}
		}
		return NULL;
		}
			struct node *findpre(int ele,struct node *head)
			{
			struct node *pre,*p;
			pre=head;
			p=head➔link;
			while(p!=null)
			{
			if(p➔data==x)
			return pre;
			else
			{
			pre=p;
			p=p➔link;
			}
			}
			return NULL;
			}

Insert	• This function is used to insert an element into a list • Before inserting an element we have to identify the position (i.e.after which element to insert) where we have to insert. • So this function accepts data, position (data) and head address.		

	void insert(int ele,int exele,struct node *head)
	{
	struct node *p,*q;
	p=malloc(sizeof(struct node));
	p➔data=ele;
	q=find(exele,head);
	p➔link=q➔link;
	q➔link=p;
	}
		void insert(int ele,int exele,struct node *head)
		{
		struct dnode *p,*q;
		p=malloc(sizeof(struct dnode));
		p➔data=ele;
		q=find(exele,head);
		p➔flink=q➔flink;
		q➔flink➔blink=p
		q➔flink=p;
		p➔blink=q;
		}
			void insert(int ele,int exele,struct node *head)
			{
			struct node *p,*q;
			p=malloc(sizeof(struct node));
			p➔data=ele;
			q=find(exele,head);
			p➔link=q➔link;
			q➔link=p;
			}

InsertStart	void insertfirst(int ele,struct node *head)
	{
	struct node *p,*q;
	q=head➔link;
	p=(struct node *)malloc(sizeof(struct node));
	p➔data=ele;
	p➔link=q;
	head➔link=p;
	}
		void insertfirst(int ele,struct node *head)
		{
		struct node *p,*q;
		q=head➔link;
		p=(struct node *)malloc(sizeof(struct node));
		p➔data=ele;
		p➔blink=NULL;
		p➔flink=q;
		head➔link=p;
		}
			void insertfirst(int ele,struct node *head)
			{
			struct node *p,*q;
			p=(struct node *)malloc(sizeof(struct node));
			p➔data=ele;
			p➔link=head;
			head =q;
			}

InsertLast	void insertlast(int ele,struct node *head)
	{
	struct node *p,*q;
	q=(struct node*)malloc(sizeof(struct node));
	q➔data=ele;
	q➔link=NULL;
	p=head;
	while(p➔link!=NULL)
	{
	p=p➔link;
	}
	p➔link=q;
	}
		void insertlast(int ele,struct node *head)
		{
		struct node *p,*q;
		q=(struct node*)malloc(sizeof(struct node));
		q➔data=ele;
		q➔flink=NULL;
		p=head➔link;
		while(p➔flink!=NULL)
		{
		p=p➔flink;
		}
		q➔blink=p;
		}
			void insertlast(int ele,struct node *head)
			{
			struct node *p,*q;
			q=(struct node*)malloc(sizeof(struct node));
			q➔data=ele;
			p=head;
			while(p➔link!=p)
			{
			p=p➔link;
			}
			q➔link=p➔link;
			p➔link=q;
			}

Operations	**Single Linked List**	**Doubly Linked List**	**Circular Linked List**
Delete	• This function is used to delete a node from the list • Before deleting a node there is a necessity to identify the previous node address so as to maintain the list. • The functions find previous is used to return the previous node address.(single and circular) • The find function is used to identify the deleting node. • It accepts the deleting node data and head address of the list.		

	void delete(int ele, struct node *head)
	{
	struct node *pre,*p;
	pre=findprevious(ele,head);
	p=find(ele,head);
	pre➔link=p➔link;
	}
		void delete(int ele, struct dnode *head)
		{
		struct dnode *p;
		p=find(ele,head);
		p➔flink➔blink=p➔blink;
		p➔blink➔flink=p➔flink;
		}
			void delete(int ele,struct node *head)
			{
			struct node *pre,*p;
			pre=findprevious(ele,head);
			p=find(ele,head);
			pre➔link=p➔link;
			}

Display	• This function used to display all the data items from the linked list • For that traverse the list from head node towards the nth node.		

	void display(struct node *head)
	{
	struct node *p;
	p=head➔link;
	while(p!=NULL)
	{
	printf("%d",p➔data);
	p=p➔link;
	}
	}
		void display(struct dnode *head)
		{
		struct dnode *p;
		p=head➔flink;
		while(p!=NULL)
		{
		printf("%d",p➔data);
		p=p➔flink;
		}
		}
			void display(struct node *head)
			{
			struct node *p;
			p=head;
			while(p!=NULL)
			{
			  if(p➔link==head)
			{
			    printf("%d",p➔data);
			    break;
			 }
			  else
			  printf("%d",p➔data);
			  p=p➔link;
			 }
			}

Search	• This function used to search an element from the linked list • For that traverse the list from head node towards the nth node.		

	void search(int ele, struct node *head)	void search(int ele, struct node *head)	void search(int ele, struct node *head)
	{	{	{
	struct node *p;	struct node *p;	struct node *p;
	p=head➔link;	p=head➔link;	p=head➔link;
	while(p!=NULL)	while(p!=NULL)	while(p!=NULL)
	{	{	{
	if(p➔data==ele)	if(p➔data==ele)	if(p➔data==ele)
	{	{	{
	printf(“%d is found”,p➔data);	printf(“%d is found”,p➔data);	printf(“%d is found”,p➔data);
	break;	break;	break;
	}	}	}
	else	else	else
	p=p➔link;	p=p➔flink;	p=p➔link;
	}	}	}
	}	}	}