Experiment 10 Date:18/11/2024

Queue Using Arrays

Aim:

Program to implement queue operations using arrays.

Algorithm:

main()

1 Start
2 Initialise size=5,front=-1,rear=-1
3 Declare queue[10],ch
4 Display Choices.
5 Read option ch.
    If ch==1 call enqueue(queue)
    If ch==2 call dequeue(queue)
    If ch==3 call display(queue)
    If ch==4 exit
6 Repeat steps 4 & 5 while ch>0&&ch<4.
7 Stop

void enqueue(*queue)

```
1 Start
2 Declare item
3 if rear==size-1
    Print Queue Overflow
  else
    Read item
    if front==-1 && rear==-1
      front=rear=0
    else
      rear=rear+1
    queue[rear]=item
4 Exit
```

void dequeue(*queue)

```
1 Start
2 if front==-1 && rear==-1
    Print Queue Underflow
  else
    if front==rear
      front=rear-1
    else
```

```
        front=front+1
    3   Exit
```

void display(*queue)

```
    1   Start
    2   Declare i
    3   if rear==-1
          Print Queue Empty
        else
          for i=front to rear ,i++
            Print queue[i]
    4   Exit
```

Program

```
#include<stdio.h>
void enqueue(int *);
void dequeue(int *);
void display(int *);
int size=5;
int front=-1,rear=-1;
void main()
{
int queue[20],ch;
do
{
printf("Queue Operations\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
printf("choose an operation:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:enqueue(queue);
break;
case 2:dequeue(queue);
break;
case 3:display(queue);
break;
case 4:printf("exit\n");
break;
default:printf("enter correct value\n");
break;
}
}while(ch!=4);
}

void enqueue(int *queue)
{
```

```
int item;
if(rear==size-1)
{
printf("Queue Overflow\n");
}
else
{
printf("enter item:\n");
scanf("%d",&item);
if(front==-1 && rear==-1)
{
front=rear=0;
}
else
{
rear=rear+1;
}
queue[rear]=item;
printf("value inserted\n");
}
printf("\n");
}

void dequeue(int *queue)
{
if(front==-1 && rear==-1)
{
printf("Queue underflow\n");
}
else
{
if(front==rear)
{
front=rear=-1;
}
else
{
front=front+1;
}
printf("value deleted\n");
}
printf("\n");
}

void display(int *queue)
{
int i;
if(rear==-1)
```

```
{
printf("queue empty\n");
}
else
{
printf("queue elements:\n");
for(i=front;i<=rear;i++)
{
printf("%d\t",queue[i]);
}
}
printf("\n");
}
```

Output

```
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
Queue underflow

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue empty

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
8
value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
```

```
4.Exit
choose an operation:
1
enter item:
7
value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
8       7
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
6
value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
5
value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
8       7       6       5
Queue Operations
```

```
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
9
value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
Queue Overflow

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
8    7    6    5    9
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
Queue Overflow

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
value deleted

Queue Operations
1.Enqueue
```

```
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
7       6       5       9
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
value deleted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
6       5       9
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
value deleted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
5       9
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
```

```
choose an operation:
2
value deleted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
value deleted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue empty

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
Queue underflow

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
4
exit
```

Experiment 11 **Date:18/11/2024**

Queue Using Arrays

Aim:

Program to implement circular queue using array.

Algorithm:

main()

1 Start
2 Initialise size=5,front=-1,rear=-1,count=0
3 Declare queue[10],ch
4 Display Choices.
5 Read option ch.
    If ch==1 call enqueue(queue)
    If ch==2 call dequeue(queue)
    If ch==3 call display(queue)
    If ch==4 exit
6 Repeat steps 4 & 5 while ch>0&&ch<4.
7 Stop

void enqueue(*queue)

1 Start
2 Declare item
3 if count==size
    Print Queue Overflow
  else
    Read item
    if front==-1 && rear==-1
      front=rear=0
    else
      rear=(rear+1)%size
    queue[rear] = item
    count=count+1
4 Exit

void dequeue(*queue)

1 Start
2 if count==0
    Print Queue Underflow
  else
    if front==rear
      front=rear-1

```
        else
          front=(front+1)%size
        count=count-1
    3   Exit
```

void display(*queue)

```
    1   Start
    2   Declare i
    3   if count==0
          Print Queue Empty
        else
          i=front
          while i !=rear
            print queue[i]
            i = (i+1) % size
          print queue[rear]
    4   Exit
```

Program

```
#include<stdio.h>
void enqueue(int *);
void dequeue(int *);
void display(int *);
int size=5;
int front=-1,rear=-1,count=0;
void main()
{
int queue[20],ch;
do
{
printf("Queue Operations\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
printf("choose an operation:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:enqueue(queue);
break;
case 2:dequeue(queue);
break;
case 3:display(queue);
break;
case 4:printf("exit\n");
break;
default:printf("enter correct value\n\n");
break;
}
```

```
}while(ch!=4);
}

void enqueue(int *queue)
{
int item;
if(count==size)
{
printf("Queue Overflow\n");
}
else
{
printf("enter item:\n");
scanf("%d",&item);
if(front==-1 && rear==-1)
{
front=rear=0;
}
else
if(front==-1&&rear==-1)
{
front=rear=0;
}
else
{
rear=(rear+1)%size;
}
queue[rear] = item;
count=count+1;
printf("Value inserted\n");
}
printf("\n");
}

void dequeue(int *queue)
{
if(count==0)
{
printf("Queue underflow\n");
}
else
{
if(front==rear)
{
front=rear=-1;
}
else
{
```

```
front=(front+1)%size;
}
count=count-1;
printf("value deleted\n");
}
printf("\n");
}

void display(int *queue)
{
int i;
if(count==0)
{
printf("queue empty\n");
}
else
{
printf("queue elements:\n");
int i=front;
while(i!=rear)
{
printf("%d\t", queue[i]);
i = (i+1) % size;
}
printf("%d\n", queue[rear]);
}
printf("\n");
}
```

Output

```
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
Queue underflow

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue empty
```

```
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
8
Value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
7
Value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
8       7

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
6
Value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
```

```
choose an operation:
1
enter item:
5
Value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
8       7       6       5

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
4
Value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
Queue Overflow

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
8       7       6       5       4

Queue Operations
1.Enqueue
```

```
2.Dequeue
3.Display
4.Exit
choose an operation:
2
value deleted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
value deleted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
6       5       4

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
8
Value inserted

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter item:
7
Value inserted
```

```
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
queue elements:
6       5       4       8       7

Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
4
exit
```

Experiment 12 **Date:20/11/2024**

Singly Linked List

Aim:

To implement the following operations on a singly linked list

i. Creation
ii. Insert a new node at front
iii. Insert an element after a particular node
iv. Insert a new node at end
v. Searching
vi. Traversal.

Algorithm:

Declare the Structure node

struct node

1.declare data, struct node* next

main()

1 Start
2 Declare ch,struct node* head = NULL
3 call createnode(head)
4 Display Choices.
5 Read option ch.
If ch==1 call insertatfront(head)
If ch==2 call insertafterkey(head)
If ch==3 call insertatlast(head)
If ch==4 call valuesearch(head)
If ch==5 call traverse(head)
If ch==6 exit
6 Repeat steps 4 & 5 while ch>0&&ch<6
7 Stop

struct node *createnode(struct node* head)

1 Start
2 Declare n,value,struct node *p
3 Read n
4 if n <= 0
Print Size Must Be Greater Than Zero
5 for i=1 to n
struct node* temp = (struct node*)malloc(sizeof(struct node));
Read value
temp->data=value
temp->next=NULL

```
    if head==NULL
      head=temp
    else
      p=head
      while p->next != NULL
        p=p->next
      p->next=temp
6   Return head
7   Exit
```

struct node *insertatfront(struct node* head)

```
1   Start
2   Declare value
3   struct node* newnode = (struct node*)malloc(sizeof(struct node));
4   Read value
5   newnode->data=value
6   newnode->next=NULL
7   if head==NULL
      newnode->next=NULL
      head=newnode
    else
      newnode->next=head;
      head=newnode;
8   Return head
9   Exit
```

struct node *insertafterkey(struct node* head)

```
1   Start
2   Declare value,key,struct node *ptr
3   struct node* newnode = (struct node*)malloc(sizeof(struct node));
4   Read value
5   newnode->data=value
6   newnode->next=NULL
7   Read key
8   if head==NULL
      newnode->next=NULL
      head=newnode
    else
      ptr=head
      while ptr != NULL
        if ptr->data == key
          newnode->next=ptr->next;
```

```
            ptr->next=newnode;
          else
            ptr=ptr->next
        if ptr==NULL
          Print Node With Key Not Exist
    9   Return head
    10  Exit
```

struct node *insertatlast(struct node* head)

```
    1   Start
    2   Declare value,struct node *ptr
    3   struct node* newnode = (struct node*)malloc(sizeof(struct node));
    4   Read value
    5   newnode->data=value
    6   newnode->next=NULL
    7   if head==NULL
          newnode->next=NULL
          head=newnode
        else
          ptr=head
          while ptr->next != NULL
            ptr=ptr->next
    8   ptr->next=newnode
    9   Return head
    10  Exit
```

void valuesearch(struct node* head)

```
    1   Start
    2   Declare value,flag=0,pos=1,struct node *ptr
    3   Read value
    4   ptr=head
    5   while ptr != NULL
          if ptr->data==value
            flag=1
            Print Value Present At Position pos
          else
            ptr=ptr->next
            pos=pos+1
    6   if flag==0
          Print Value Not Found
    7   Exit
```

void traverse(struct node* head)

1 Start
2 Declare struct node *ptr
3 ptr=head
4 if head==NULL
  Print List Empty
  else
  while ptr != NULL
  Print ptr->data
  ptr=ptr->next
  Print Null
5 Exit

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node* createnode(struct node* head)
{
struct node *p;
int value,n;
printf("enter size \n");
scanf("%d",&n);
if(n <= 0) {
printf("List size must be greater than 0.\n");
return 0;
}
for(int i=1;i<=n;i++)
{
struct node* temp = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
temp->data=value;
temp->next=NULL;
if (head==NULL)
{
head=temp;
}
else
{
p=head;
```

```
while(p->next!=NULL)
{
p=p->next;
}
p->next=temp;
}
}
return head;
}
struct node* insertatfront(struct node* head);
struct node* insertatlast(struct node* head);
struct node* insertafterkey(struct node* head);
void traverse(struct node* head);
void valuesearch(struct node* head);
void main()
{
int ch,data,pos,val;
struct node* head = NULL;
printf("Creating a linked list:\n");
head=createnode(head);
do
{
printf("Linked List Operations\n1.Insert Node At Front\n2.Insert After Particular
Node\n3.Insert Node At Last\n4.Searching\n5.Traversal\n6.Exit\n");
printf("choose an operation:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
{
head=insertatfront(head);
printf("value inserted at front\n");
break;
}
case 2:
{
head=insertafterkey(head);
break;
}
case 3:
{
head=insertatlast(head);
printf("value inserted at last\n");
break;
}
case 4:
{
printf("searching\n");
```

```
valuesearch(head);
break;
}
case 5:
{
printf("traversing\n");
traverse(head);
break;
}
case 6:printf("exit\n");
break;
default:printf("enter correct value\n\n");
break;
}
}while(ch!=6);
}

struct node* insertatfront(struct node* head)
{
int value;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->next=NULL;
if(head==NULL)
{
newnode->next=NULL;
head=newnode;
}
else
{
newnode->next=head;
head=newnode;
}
return head;
}

struct node* insertatlast(struct node* head)
{
int value;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->next=NULL;
struct node *ptr;
if(head==NULL)
```

```
{
newnode->next=NULL;
head=newnode;
}
else
{
ptr=head;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
}
ptr->next=newnode;
return head;
}

struct node* insertafterkey(struct node* head)
{
int value,key;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->next=NULL;
printf("enter key\n");
scanf("%d",&key);
struct node *ptr;
if(head==NULL)
{
newnode->next=NULL;
head=newnode;
}
else
{
ptr=head;
while (ptr != NULL)
{
if (ptr->data == key)
{
newnode->next=ptr->next;
ptr->next=newnode;
printf("Node inserted after key\n");
return head;
}
else
{
ptr = ptr->next;
}
```

```
}
if (ptr == NULL)
{
printf("Node with key does not exist\n");
}
}
return head;
}

void traverse(struct node* head)
{
struct node *ptr;
ptr=head;
if(head==NULL)
{
printf("list empty\n");
}
else
{
while(ptr!=NULL)
{
printf("%d->",ptr->data);
ptr=ptr->next;
}
printf("NULL\n");
}
}

void valuesearch(struct node* head)
{
int val;
printf("enter value to search\n");
scanf("%d",&val);
struct node *ptr;
ptr=head;
int flag=0,pos=1;
while(ptr!=NULL)
{
if(ptr->data==val)
{
flag=1;
printf("Item Present at position %d\n",pos);
break;
}
else
{
ptr=ptr->next;
pos=pos+1;
```

```
}
}
if(flag==0)
{
printf("Item Not Found\n");
}
}
```

Output

```
Creating a linked list:
enter size
5
enter value to insert
4
enter value to insert
5
enter value to insert
6
enter value to insert
7
enter value to insert
8
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
5
traversing
4->5->6->7->8->NULL
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
1
enter value to insert
3
value inserted at front
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
```

3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
5
traversing
3->4->5->6->7->8->NULL
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
3
enter value to insert
9
value inserted at last
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
5
traversing
3->4->5->6->7->8->9->NULL
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
2
enter value to insert
7
enter key
5
Node inserted after key
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node

```
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
5
traversing
3->4->5->7->6->7->8->9->NULL
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
2
enter value to insert
10
enter key
10
Node with key does not exist
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
4
searching
enter value to search
8
Item Present at position 7
Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
4
searching
enter value to search
12
Item Not Found
```

Linked List Operations
1.Insert Node At Front
2.Insert After Particular Node
3.Insert Node At Last
4.Searching
5.Traversal
6.Exit
choose an operation:
6
exit

Experiment 13 **Date:02/12/2024**

Singly Linked List

Aim:

To implement the following operations on a singly linked list

i. Creation
ii. Deletion from beginning
iii. Deletion from the end
iv. Deletion from particular location
v. Traversal.

Algorithm:

Declare the Structure node

struct node

1.declare data, struct node* next

main()

1 Start
2 Declare ch,struct node* head = NULL
3 call createnode(head)
4 Display Choices.
5 Read option ch.
If ch==1 call deleteatfront(head)
If ch==2 call deleteatlast(head)
If ch==3 call deleteatpos(head)
If ch==4 call traverse(head)
If ch==5 exit
6 Repeat steps 4 & 5 while ch>0&&ch<5
7 Stop

struct node *createnode(struct node* head)

1 Start
2 Declare n,value,struct node *p
3 Read n
4 if n <= 0
Print Size Must Be Greater Than Zero
5 for i=1 to n
struct node* temp = (struct node*)malloc(sizeof(struct node));
Read value
temp->data=value
temp->next=NULL
if head==NULL
head=temp

```
        else
          p=head
          while p->next != NULL
            p=p->next
          p->next=temp
    6   Return head
    7   Exit
```

struct node *deleteatfront(struct node* head)

```
    1   Start
    2   Declare struct node *ptr
    3   if head==NULL
          Print Linked List Underflow
        else
          ptr=head
          head=ptr->next
          free(ptr)
    4   Return head
    5   Exit
```

struct node *deleteatlast(struct node* head)

```
    1   Start
    2   Decalre struct node *ptr,struct node *ptr1
    3   if head==NULL
          Print Linked List Underflow
        else
          if ptr->next == NULL
            head=NULL
            free(head)
          else
            ptr=head
            while ptr->next != NULL
              ptr1=ptr
              ptr=ptr->next
            ptr1->next=NULL
            free(ptr)
    4   Return head
    5   Exit
```

struct node *deleteatpos(struct node* head)

```
1  Start
2  Declare i,pos,struct node *ptr,struct node *ptr1
3  Read pos
4  if head==NULL
     Print Linked List Underflow
5  if pos==1
     ptr=head
     head=ptr->next
     free(ptr)
   else
     ptr=head
     for i=1 to i<pos-1 && ptr!==NULL,i++
       ptr=ptr->next
     if ptr==NULL||ptr->next==NULL
       Print Position Out Of Range
     ptr1 = ptr->next;
     ptr->next = ptr1->next;
     free(ptr1);
6  Return head
7  Exit
```

void traverse(struct node* head)

```
1  Start
2  Declare struct node *ptr
3  ptr=head
4  if head==NULL
     Print List Empty
   else
     while ptr != NULL
       Print ptr->data
       ptr=ptr->next
     Print Null
5  Exit
```

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node* createnode(struct node* head)
{
struct node *p;
int value,n;
printf("enter size \n");
scanf("%d",&n);
if(n <= 0) {
printf("List size must be greater than 0.\n");
return 0;
}
for(int i=1;i<=n;i++)
{
struct node* temp = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
temp->data=value;
temp->next=NULL;
if (head==NULL)
{
head=temp;
}
else
{
p=head;
while(p->next!=NULL)
{
p=p->next;
}
p->next=temp;
}
}
return head;
}
struct node* deleteatfront(struct node* head);
struct node* deleteatlast(struct node* head);
struct node* deleteatpos(struct node* head);
void traverse(struct node* head);
void main()
{
```

```
int ch,data,pos,val;
struct node *head=NULL;
printf("Creating a linked list:\n");
head=createnode(head);
do
{
printf("Linked List Operations\n1.Delete From Front\n2.Delete From Last\n3.Delete From Particular Position\n4.Traversal\n5.Exit\n");
printf("choose an operation:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
{
head=deleteatfront(head);
printf("value deleted from front\n");
break;
}
case 2:
{
head=deleteatlast(head);
printf("value deleted from last\n");
break;
}
case 3:
{
head=deleteatpos(head);
break;
}
case 4:
{
printf("traversing\n");
traverse(head);
break;
}
case 5:printf("exit\n");
break;
default:printf("enter correct value\n\n");
break;
}
}while(ch!=5);
}

struct node* deleteatfront(struct node* head)
{
if(head==NULL)
{
printf("linked list underflow\n");
```

```
}
else
{
struct node *ptr;
ptr=head;
head=ptr->next;
free(ptr);
}
return head;
}

struct node* deleteatlast(struct node* head)
{
if(head==NULL)
{
printf("linked list underflow\n");
}
else
{
if(head->next==NULL)
{
head=NULL;
free(head);
}
else
{
struct node *ptr;
struct node *ptr1;
ptr=head;
while(ptr->next!=NULL)
{
ptr1=ptr;
ptr=ptr->next;
}
ptr1->next=NULL;
free(ptr);
}
}
return head;
}

struct node* deleteatpos(struct node* head)
{
struct node *ptr;
struct node *ptr1;
int i,pos;
printf("enter position\n");
scanf("%d",&pos);
```

```
if (head == NULL)
{
printf("Linked list underflow\n");
return head;
}
if (pos==1)
{
ptr=head;
head=ptr->next;
free(ptr);
printf("Value deleted from position\n");
}
else
{
ptr = head;
for(i=1;i<pos-1&&ptr!=NULL;i++)
{
ptr = ptr->next;
}
if (ptr==NULL||ptr->next==NULL)
{
printf("Position out of range\n");
return head;
}
ptr1 = ptr->next;
ptr->next = ptr1->next;
free(ptr1);
printf("Value deleted from position\n");
}
return head;
}

void traverse(struct node* head)
{
struct node *ptr;
ptr=head;
if(head==NULL)
{
printf("list empty\n");
}
else
{
while(ptr!=NULL)
{
printf("%d->",ptr->data);
ptr=ptr->next;
}
printf("NULL\n");
```

```
}
}
```

Output

```
Creating a linked list:
enter size
7
enter value to insert
2
enter value to insert
3
enter value to insert
4
enter value to insert
5
enter value to insert
6
enter value to insert
7
enter value to insert
8
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
choose an operation:
4
traversing
2->3->4->5->6->7->8->NULL
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
choose an operation:
1
value deleted from front
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
choose an operation:
4
```

```
traversing
3->4->5->6->7->8->NULL
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
choose an operation:
2
value deleted from last
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
choose an operation:
4
traversing
3->4->5->6->7->NULL
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
choose an operation:
3
enter position
3
Value deleted from position
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
choose an operation:
4
traversing
3->4->6->7->NULL
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
```

choose an operation:
3
enter position
6
Position out of range
Linked List Operations
1.Delete From Front
2.Delete From Last
3.Delete From Particular Position
4.Traversal
5.Exit
choose an operation:
5
exit

Experiment 14 Date:02/12/2024

Stack Using Singly Linked List

Aim:

To implement a menu driven program to perform following stack operations using linked list

i. Push
ii. pop
iii.Traversal

Algorithm:

Declare the Structure node

struct node

1.declare data, struct node* next

main()

1 Start
2 Declare ch,struct node* head = NULL
3 Display Choices.
4 Read option ch.
    If ch==1 call push(head)
    If ch==2 call pop(head)
    If ch==3 call display(head)
    If ch==4 exit
5 Repeat steps 3 & 4 while ch>0&&ch<4
6 Stop

struct node *push(struct node* head)

1 Start
2 Declare value
3 struct node* newnode = (struct node*)malloc(sizeof(struct node));
4 Read value
5 newnode->data=value
6 newnode->next=NULL
7 if head==NULL
    newnode->next=NULL
    head=newnode
  else
    newnode->next=head;
    head=newnode;
8 Return head
9 Exit

struct node *pop(struct node* head)

1 Start
2 Declare struct node *ptr
3 if head==NULL
  Print Stack Underflow
  else
  ptr=head
  head=ptr->next
  free(ptr)
4 Return head
5 Exit

void display(struct node* head)

1 Start
2 Declare struct node *ptr
3 ptr=head
4 if head==NULL
  Print Stack Empty
  else
  while ptr != NULL
  Print ptr->data
  ptr=ptr->next
  Print Null
5 Exit

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node* push(struct node* head);
struct node* pop(struct node* head);
void display(struct node* head);
void main()
{
int ch;
struct node* head = NULL;
do
{
printf("Stack Operations\n1.Push\n2.Pop\n3.Display\n4.Exit\n");
printf("choose an operation:\n");
```

```
scanf("%d",&ch);
switch(ch)
{
case 1:
{
head=push(head);
break;
}
case 2:
{
head=pop(head);
break;
}
case 3:
{
printf("display\n");
display(head);
break;
}
case 4:printf("exit\n");
break;
default:printf("enter correct value\n\n");
break;
}
}while(ch!=4);
}

struct node* push(struct node* head)
{
int value;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->next=NULL;
struct node *ptr;
if(head==NULL)
{
newnode->next=NULL;
head=newnode;
}
else
{
newnode->next=head;
head=newnode;
}
printf("value pushed\n");
return head;
```

```
}

struct node* pop(struct node* head)
{
if(head==NULL)
{
printf("underflow\n");
}
else
{
struct node *ptr;
ptr=head;
head=ptr->next;
free(ptr);
printf("value poped\n");
}
return head;
}

void display(struct node* head)
{
struct node *ptr;
ptr=head;
if(head==NULL)
{
printf("stack empty\n");
}
else
{
while(ptr!=NULL)
{
printf("%d->",ptr->data);
ptr=ptr->next;
}
printf("NULL\n");
}
}
```

Output

```
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
2
underflow
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
3
display
stack empty
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
1
enter value to insert
4
value pushed
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
1
enter value to insert
5
value pushed
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
3
display
5->4->NULL
```

```
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
1
enter value to insert
6
value pushed
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
1
enter value to insert
7
value pushed
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
3
display
7->6->5->4->NULL
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
1
enter value to insert
8
value pushed
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
1
enter value to insert
9
```

```
value pushed
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
3
display
9->8->7->6->5->4->NULL
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
2
value poped
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
3
display
8->7->6->5->4->NULL
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
2
value poped
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
3
display
7->6->5->4->NULL
Stack Operations
1.Push
2.Pop
3.Display
```

```
4.Exit
choose an operation:
2
value poped
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
3
display
6->5->4->NULL
Stack Operations
1.Push
2.Pop
3.Display
4.Exit
choose an operation:
4
Exit
```

Experiment 15 **Date:02/12/2024**

Queue Using Singly Linked List

Aim:

To implement a menu driven program to perform following queue operations using linked list

i. Enqueue
ii. Dequeue
iii.Traversal

Algorithm:

Declare the Structure node

struct node

1.declare data, struct node* next

main()

1 Start
2 Declare ch,struct node* head = NULL
3 Display Choices.
4 Read option ch.
If ch==1 call enqueue(head)
If ch==2 call dequeue(head)
If ch==3 call display(head)
If ch==4 exit
5 Repeat steps 3 & 4 while ch>0&&ch<4
6 Stop

struct node *enqueue(struct node* head)

1 Start
2 Declare value,struct node *ptr
3 struct node* newnode = (struct node*)malloc(sizeof(struct node));
4 Read value
5 newnode->data=value
6 newnode->next=NULL
7 if head==NULL
newnode->next=NULL
head=newnode
else
ptr=head
while ptr->next != NULL
ptr=ptr->next
8 ptr->next=newnode
9 Return head
10 Exit

struct node *dequeue(struct node* head)

1 Start
2 Declare struct node *ptr
3 if head==NULL
  Print Queue Underflow
  else
  ptr=head
  head=ptr->next
  free(ptr)
4 Return head
5 Exit

void display(struct node* head)

1 Start
2 Declare struct node *ptr
3 ptr=head
4 if head==NULL
  Print Stack Empty
  else
  while ptr != NULL
  Print ptr->data
  ptr=ptr->next
  Print Null
5 Exit

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node* enqueue(struct node* head);
struct node* dequeue(struct node* head);
void display(struct node* head);
void main()
{
int ch;
struct node* head = NULL;
do
{
printf("Queue Operations\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
printf("choose an operation:\n");
```

```
scanf("%d",&ch);
switch(ch)
{
case 1:
{
head=enqueue(head);
break;
}
case 2:
{
head=dequeue(head);
break;
}
case 3:
{
printf("display\n");
display(head);
break;
}
case 4:printf("exit\n");
break;
default:printf("enter correct value\n\n");
break;
}
}while(ch!=4);
}

struct node* enqueue(struct node* head)
{
int value;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->next=NULL;
struct node *ptr;
if(head==NULL)
{
newnode->next=NULL;
head=newnode;
}
else
{
ptr=head;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
```

```
}
ptr->next=newnode;
printf("value inserted\n");
return head;
}

struct node* dequeue(struct node* head)
{
if(head==NULL)
{
printf("underflow\n");
}
else
{
struct node *ptr;
ptr=head;
head=ptr->next;
free(ptr);
printf("value deleted\n");
}
return head;
}

void display(struct node* head)
{
struct node *ptr;
ptr=head;
if(head==NULL)
{
printf("stack empty\n");
}
else
{
while(ptr!=NULL)
{
printf("%d->",ptr->data);
ptr=ptr->next;
}
printf("NULL\n");
}
}
```

Output

```
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
underflow
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
display
stack empty
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter value to insert
2
value inserted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter value to insert
3
value inserted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
display
2->3->NULL
```

```
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter value to insert
4
value inserted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter value to insert
5
value inserted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
display
2->3->4->5->NULL
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter value to insert
6
value inserted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
1
enter value to insert
7
```

```
value inserted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
display
2->3->4->5->6->7->NULL
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
value deleted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
display
3->4->5->6->7->NULL
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
2
value deleted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
display
4->5->6->7->NULL
Queue Operations
1.Enqueue
2.Dequeue
3.Display
```

```
4.Exit
choose an operation:
2
value deleted
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
3
display
5->6->7->NULL
Queue Operations
1.Enqueue
2.Dequeue
3.Display
4.Exit
choose an operation:
4
exit
```

Experiment 16 Date:04/12/2024

Doubly Linked List

Aim:

To implement the following operations on a Doubly linked list.

i. Creation
ii. Count the number of nodes
iii. Searching
iv.Traversal

Algorithm:

Declare the Structure node

struct node

1.declare data, struct node* prev,struct node *next

main()

1 Start
2 Declare ch,struct node* head = NULL
3 call createnode(head)
4 Display Choices.
5 Read option ch.
If ch==1 call noofnode(head)
If ch==2 call searchnode(head)
If ch==3 call traverse(head)
If ch==4 exit
6 Repeat steps 4 & 5 while ch>0&&ch<5
7 Stop

struct node *createnode(struct node* head)

1 Start
2 Declare n,value,struct node *p
3 Read n
4 if n <= 0
Print Size Must Be Greater Than Zero
5 for i=1 to n
struct node* temp = (struct node*)malloc(sizeof(struct node));
Read value
temp->data=value
temp->prev=NULL
temp->next=NULL
if head==NULL
head=temp

```
        else
          p=head
          while p->next != NULL
            p=p->next
          p->next=temp
          temp->prev=p
    6   Return head
    7   Exit
```

struct node *noofnode(struct node* head)

```
    1   Start
    2   Declare count=0,struct node *ptr
    3   ptr=head
    4   while ptr != NULL
          ptr=ptr->next
          count=count+1
    5   Print count
    6   Exit
```

void searchnode(struct node* head)

```
    1   Start
    2   Declare value,flag=0,pos=1,struct node *ptr
    3   Read value
    4   ptr=head
    5   while ptr != NULL
          if ptr->data==value
            flag=1
            Print Value Present At Position pos
          else
            ptr=ptr->next
            pos=pos+1
    6   if flag==0
          Print Value Not Found
    7   Exit
```

void traverse(struct node* head)

```
    1   Start
    2   Declare struct node *ptr
    3   ptr=head
    4   if head==NULL
          Print List Empty
```

```
        else
          while ptr != NULL
            Print ptr->data
            ptr=ptr->next
          Print Null
    5   Exit
```

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
struct node *prev;
};
struct node* createnode(struct node* head)
{
struct node *p;
int value,n;
printf("enter size \n");
scanf("%d",&n);
if(n <= 0) {
printf("List size must be greater than 0.\n");
return 0;
}
for(int i=1;i<=n;i++)
{
struct node* temp = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
temp->data=value;
temp->prev=NULL;
temp->next=NULL;
if (head==NULL)
{
head=temp;
}
else
{
p=head;
while(p->next!=NULL)
{
p=p->next;
}
p->next=temp;
```

```
temp->prev=p;
}
}
return head;
}
void searchnode(struct node* head);
void noofnodes(struct node* head);
void traverse(struct node* head);
void main()
{
int ch,val;
struct node *head=NULL;
printf("Creating a linked list:\n");
head=createnode(head);
do
{
printf("Linked List Operations\n1.Count No Of Nodes\n2.Searching a node\n3.Traversal\n4.Exit\n");
printf("choose an operation:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
{
printf("Node Count\n");
noofnodes(head);
break;
}
case 2:
{
printf("searching\n");
searchnode(head);
break;
}
case 3:
{
printf("traversing\n");
traverse(head);
break;
}
case 4:printf("exit\n");
break;
default:printf("enter correct value\n\n");
break;
}
}while(ch!=4);
}
```

```
void searchnode(struct node* head)
{
struct node *ptr;
ptr=head;
int flag=0,pos=1,val;
printf("enter value to search\n");
scanf("%d",&val);
while(ptr!=NULL)
{
if(ptr->data==val)
{
flag=1;
printf("Item Present at position %d\n",pos);
break;
}
else
{
ptr=ptr->next;
pos=pos+1;
}
}
if(flag==0)
{
printf("Item Not Found\n");
}
}

void noofnodes(struct node* head)
{
struct node *ptr;
ptr=head;
int count=0;
while(ptr!=NULL)
{
ptr=ptr->next;
count=count+1;
}
printf("No of nodes is %d\n",count);
}

void traverse(struct node* head)
{
struct node *ptr;
ptr=head;
if(head==NULL)
{
printf("list empty\n");
}
```

```
else
{
while(ptr!=NULL)
{
printf("%d<->",ptr->data);
ptr=ptr->next;
}
printf("NULL\n");
}
}
```

Output

```
Creating a linked list:
enter size
6
enter value to insert
2
enter value to insert
3
enter value to insert
4
enter value to insert
5
enter value to insert
6
enter value to insert
7
Linked List Operations
1.Count No Of Nodes
2.Searching a node
3.Traversal
4.Exit
choose an operation:
3
traversing
2<->3<->4<->5<->6<->7<->NULL
Linked List Operations
1.Count No Of Nodes
2.Searching a node
3.Traversal
4.Exit
choose an operation:
1
Node Count
No of nodes is 6
Linked List Operations
1.Count No Of Nodes
2.Searching a node
```

3.Traversal
4.Exit
choose an operation:
2
searching
enter value to search
7
Item Present at position 6
Linked List Operations
1.Count No Of Nodes
2.Searching a node
3.Traversal
4.Exit
choose an operation:
2
searching
enter value to search
4
Item Present at position 3
Linked List Operations
1.Count No Of Nodes
2.Searching a node
3.Traversal
4.Exit
choose an operation:
2
searching
enter value to search
10
Item Not Found
Linked List Operations
1.Count No Of Nodes
2.Searching a node
3.Traversal
4.Exit
choose an operation:
4
exit

Experiment 17 **Date:04/12/2024**

Doubly Linked List

Aim:

To implement the following operations on a Doubly linked list.

i. Creation
ii. Insert a node at first position
iii. Insert a node at last
iv. Delete a node from the first position
v. Delete a node from last
vi.Traversal

Algorithm:

Declare the Structure node

struct node

1.declare data, struct node* prev,struct node *next

main()

1 Start
2 Declare ch,struct node* head = NULL
3 call createnode(head)
4 Display Choices.
5 Read option ch.
If ch==1 call insertatfront(head)
If ch==2 call insertatlast(head)
If ch==3 call deleteatfront(head)
If ch==4 call deleteatlast(head)
If ch==5 call traverse(head)
If ch==6 exit
6 Repeat steps 4 & 5 while ch>0&&ch<6
7 Stop

struct node *createnode(struct node* head)

1 Start
2 Declare n,value,struct node *p
3 Read n
4 if n <= 0
Print Size Must Be Greater Than Zero
5 for i=1 to n
struct node* temp = (struct node*)malloc(sizeof(struct node));
Read value
temp->data=value
temp->prev=NULL
temp->next=NULL

```
    if head==NULL
      head=temp
    else
      p=head
      while p->next != NULL
        p=p->next
      p->next=temp
      temp->prev=p
6   Return head
7   Exit
```

struct node *insertatfront(struct node* head)

```
1   Start
2   Declare value
3   struct node* newnode = (struct node*)malloc(sizeof(struct node));
4   Read value
5   newnode->data=value
6   newnode->prev=NULL
7   newnode->next=NULL
8   if head==NULL
      newnode->prev=NULL
      newnode->next=NULL
      head=newnode
    else
      newnode->prev=NULL
      newnode->next=head;
      head=newnode;
9   Return head
10  Exit
```

struct node *insertatlast(struct node* head)

```
1   Start
2   Declare value,struct node *ptr
3   struct node* newnode = (struct node*)malloc(sizeof(struct node));
4   Read value
5   newnode->data=value
6   newnode->prev=NULL
7   newnode->next=NULL
8   if head==NULL
      newnode->next=NULL
      newnode->prev=NULL
```

```
      head=newnode
    else
      ptr=head
      while ptr->next != NULL
        ptr=ptr->next
9   ptr->next=newnode
10   newnode->prev=ptr
11   Return head
12   Exit
```

struct node *deleteatfront(struct node* head)

```
1   Start
2   Declare struct node *ptr
3   if head==NULL
      Print Linked List Underflow
    else
      ptr=head
      head=ptr->next
      head->prev=NULL
      free(ptr)
4   Return head
5   Exit
```

struct node *deleteatlast(struct node* head)

```
1   Start
2   Decalre struct node *ptr,struct node *ptr1
3   if head==NULL
      Print Linked List Underflow
    else
      if ptr->next == NULL
        head=NULL
        free(head)
      else
        ptr=head
        while ptr->next != NULL
          ptr1=ptr
          ptr=ptr->next
        ptr1->next=NULL
        ptr->prev=NULL;
        free(ptr)
4   Return head
5   Exit
```

void traverse(struct node* head)

1 Start
2 Declare struct node *ptr
3 ptr=head
4 if head==NULL
  Print List Empty

  else
  while ptr != NULL
  Print ptr->data
  ptr=ptr->next
  Print Null
5 Exit

Program

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
struct node *prev;
};
struct node* createnode(struct node* head)
{
struct node *p;
int value,n;
printf("enter size \n");
scanf("%d",&n);
if(n <= 0) {
printf("List size must be greater than 0.\n");
return 0;
}
for(int i=1;i<=n;i++)
{
struct node* temp = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
temp->data=value;
temp->prev=NULL;
temp->next=NULL;
if (head==NULL)
{
head=temp;
}
```

```
else
{
p=head;
while(p->next!=NULL)
{
p=p->next;
}
p->next=temp;
temp->prev=p;
}
}
return head;
}

struct node* insertatfront(struct node* head);
struct node* insertatlast(struct node* head);
struct node* deleteatfront(struct node* head);
struct node* deleteatlast(struct node* head);
void traverse(struct node* head);
void main()
{
int ch,data;
struct node *head=NULL;
printf("Creating a linked list:\n");
head=createnode(head);
do
{
printf("Linked List Operations\n1.Insert Node At Front\n2.Insert Node At Last\n3.Delete Node At Front\n4.Delete Node At Last\n5.Traversal\n6.Exit\n");
printf("choose an operation:\n");
scanf("%d",&ch);
switch(ch)
{
case 1:
{
head=insertatfront(head);
printf("value inserted at front\n");
break;
}
case 2:
{
head=insertatlast(head);
printf("value inserted at last\n");
break;
}
case 3:
{
head=deleteatfront(head);
```

```
printf("value deleted from front\n");
break;
}
case 4:
{
head=deleteatlast(head);
printf("value deleted from last\n");
break;
}
case 5:
{
printf("traversing\n");
traverse(head);
break;
}
case 6:printf("exit\n");
break;
default:printf("enter correct value\n\n");
break;
}
}while(ch!=6);
}

struct node* insertatfront(struct node* head)
{
int value;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->prev=NULL;
newnode->next=NULL;
if(head==NULL)
{
newnode->next=NULL;
newnode->prev=NULL;
head=newnode;
}
else
{
newnode->next=head;
newnode->prev=NULL;
head=newnode;
}
return head;
}

struct node* insertatlast(struct node* head)
```

```
{
int value;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
printf("enter value to insert\n");
scanf("%d",&value);
newnode->data=value;
newnode->prev=NULL;
newnode->next=NULL;
struct node *ptr;
if(head==NULL)
{
newnode->next=NULL;
newnode->prev=NULL;
head=newnode;
}
else
{
ptr=head;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
}
ptr->next=newnode;
newnode->prev=ptr;
return head;
}

struct node* deleteatfront(struct node* head)
{
if(head==NULL)
{
printf("linked list underflow\n");
}
else
{
struct node *ptr;
ptr=head;
head=ptr->next;
head->prev=NULL;
free(ptr);
}
return head;
}

struct node* deleteatlast(struct node* head)
{
if(head==NULL)
```

```
{
printf("linked list underflow\n");
}
else
{
if(head->next==NULL)
{
head=NULL;
free(head);
}
else
{
struct node *ptr;
struct node *ptr1;
ptr=head;
while(ptr->next!=NULL)
{
ptr1=ptr;
ptr=ptr->next;
}
ptr1->next=NULL;
ptr->prev=NULL;
free(ptr);
}
}
return head;
}

void traverse(struct node* head)
{
struct node *ptr;
ptr=head;
if(head==NULL)
{
printf("list empty\n");
}
else
{
while(ptr!=NULL)
{
printf("%d<->",ptr->data);
ptr=ptr->next;
}
printf("NULL\n");
}
}
```

Output

```
Creating a linked list:
enter size
6
enter value to insert
6
enter value to insert
7
enter value to insert
8
enter value to insert
9
enter value to insert
10
enter value to insert
11
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
5
traversing
6<->7<->8<->9<->10<->11<->NULL
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
1
enter value to insert
5
value inserted at front
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
```

```
5
traversing
5<->6<->7<->8<->9<->10<->11<->NULL
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
2
enter value to insert
12
value inserted at last
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
5
traversing
5<->6<->7<->8<->9<->10<->11<->12<->NULL
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
3
value deleted from front
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
5
traversing
6<->7<->8<->9<->10<->11<->12<->NULL
Linked List Operations
```

```
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
4
value deleted from last
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
5
traversing
6<->7<->8<->9<->10<->11<->NULL
Linked List Operations
1.Insert Node At Front
2.Insert Node At Last
3.Delete Node At Front
4.Delete Node At Last
5.Traversal
6.Exit
choose an operation:
6
exit
```