1 Implement Stack using array

```
#include<stdio.h>
#define STACK_SIZE 5
int top = -1;
int s[10];
int item;
void push;
{
if (top == STACK_SIZE - 1)
{
printf ("stack overflow\n");
return;
top= top+1;
s[top] = item;
}
int pop()
{
if (top == -1)
return -1;
return s[top--];
}
void display()
{
int i;
int (top == -1)
{
printf ("stack is empty\n");
return;
}
```

```
printf ("contents of the stack \n");
for(i = top ; i >=0 ; i--)
{
printf ("%d \n", s[i]);
}
}
void main()
{
int item_deleted;
int choice;
for(;;)
{
printf ("\n 1. push \n 2. pop \n 3. display \n
        4. exit \n");
printf ("enter the choice);
switch (choice)
{
case 1: printf ("enter the item to be inserted \n");
        scanf ("%d", &choice);
        push();
        break;
case 2 : item_deleted = pop();
        if (item deleted == -1)
        printf ("stack is empty \n");
        else
        printf ("item deleted is %d \n", item_delete
        break;
case 3 : display (top, s);
default exit (0);
```

}
}

output :

1. push
2. pop.
3. display
4. exit

enter the choice
1
enter the item to be inserted
12.
enter the choice :
enter the item to be inserted : 24

Enter the choice : 3
The elements of stack.
24
12
Enter the choice : 4
Exit point.

```
        1.PUSH
        2.POP
        3.DISPLAY
        4.EXIT
Enter the Choice:1
Enter a value to be pushed:12

Enter the Choice:1
Enter a value to be pushed:24

Enter the Choice:1
Enter a value to be pushed:98

Enter the Choice:3

The elements in STACK

98
24
12
 Press Next Choice
 Enter the Choice:2

        The popped elements is 98
 Enter the Choice:3

 The elements in STACK

 24
 12
  Press Next Choice
  Enter the Choice:4

          EXIT POINT
```

IBM19CS156

5/10/2020

2. Wap to convert a given valid parenthesized infix arthimatic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) & /(divide).

```
#include<stdio.h>
#include<string.h>
int F (char symbol)
{
switch (symbol)
{
case '+' :
case '-' : return 2;
case '*' :
case '/' : return 4;
case '^' :
case '$' : return 5;
case '(' : return 0;
case '#' : return -1;
default : return 8;
}
} int G (char symbol) {
Switch (Symbol)
{
case '+' :
case '-' : return 1;
case '*' :
case '/' : return 7;
```

```
case '^' :
case '$' : return 6;
case '(' : return 9;
case ')' : return 0;
default : return 7 ;
    }
}
Void infix_postfix (char infix[] , char postfix[])
{
int top, i, j ;
char s[30], symbol ;
top --1 ;
s [++top] = '#' ;
j = 0 ;
for ( i = 0 ; i < strlen (infix) ; i++)
{
symbol = infix [i] ;
while (F [s [top]] > G (symbol))
{
postfix [j] = s [top--] ;
 j++ ;
}
if (F(s [top]) != G (symbol))
{
   s [top++] = symbol ;
else
top - - ;
}
```

```
while (s[top] != '#')
{
postfix[j++] = s[top--];
}
postfix[j] = '\0';
}
Void main()
{
Char infix[20];
char postfix[20];
printf("enter valid infix expression\n")
scanf("%s", infix);
infix_postfix(infix, postfix);
printf("the postfix expresson is\n");
printf("%s\n", postfix);
}
```

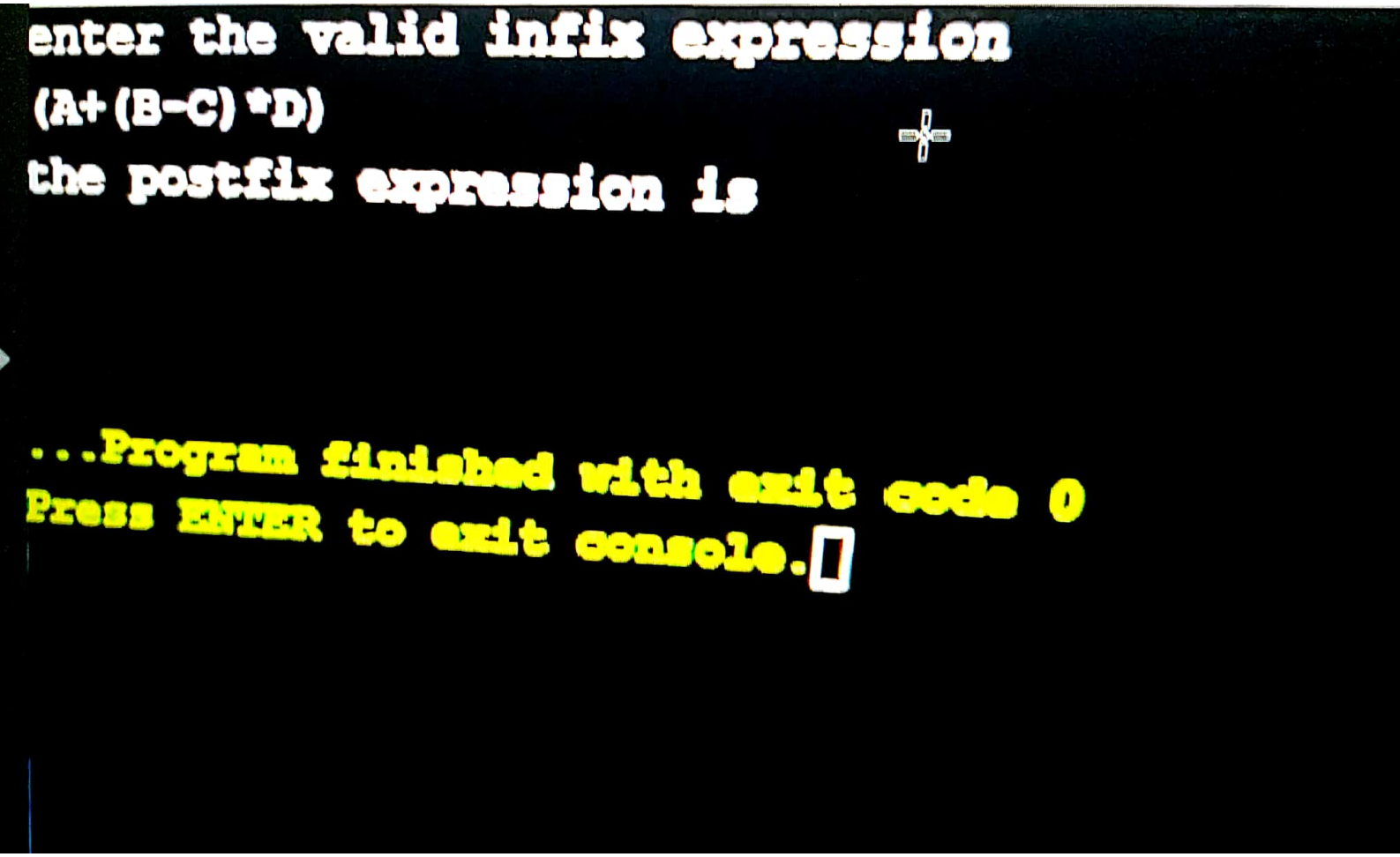
Output:

Enter valid infix expression

(A+(B-C)*D)

the post fix expression is

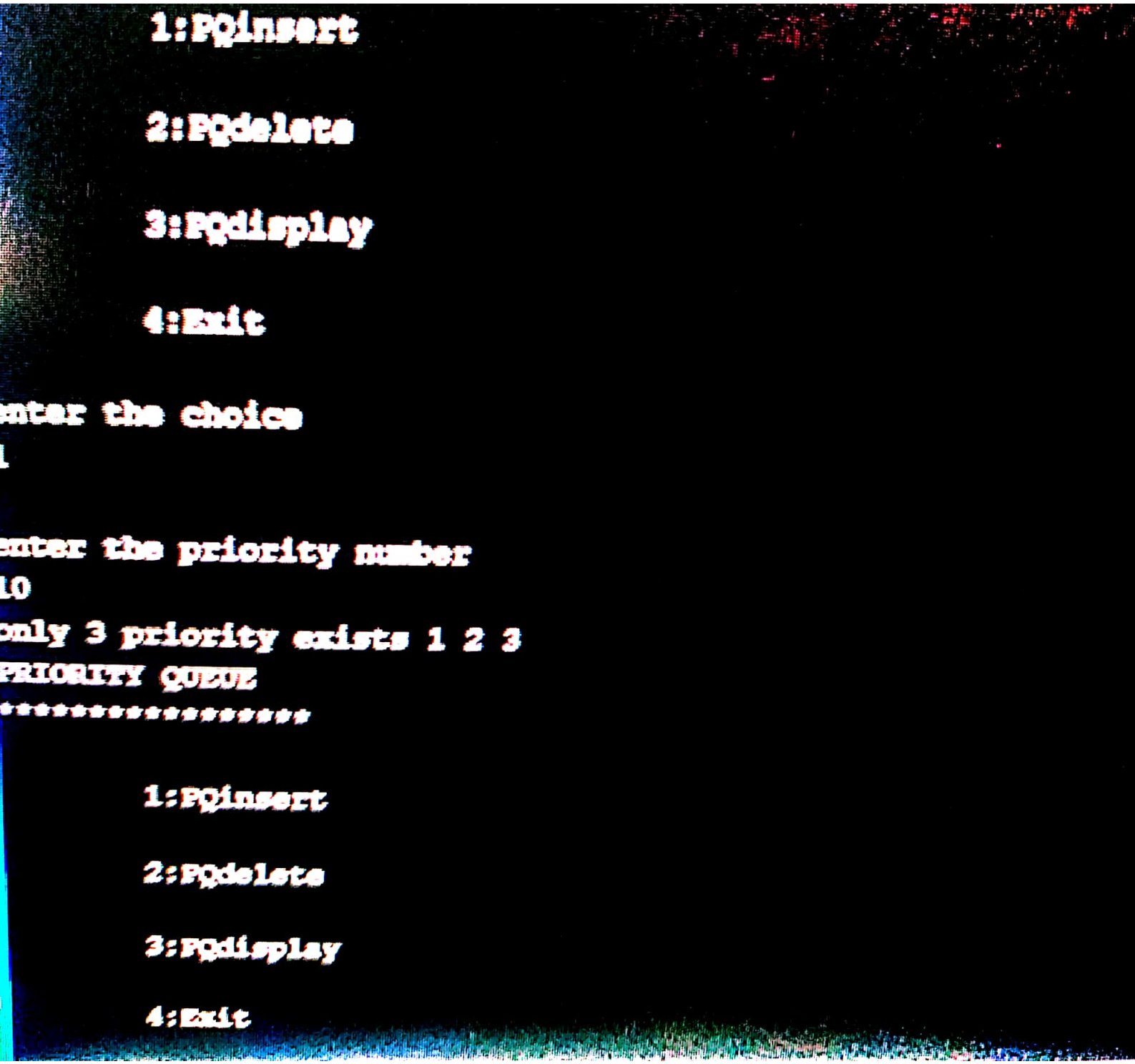
ABC-D*f.



ab:

3) Linear queue
a) insert & delete c) display

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int item, front=0, rear=-1, q[10];
void insertrear()
{
if (count == qu.size+1)
{
printf ("queue overflow");
return;
}
rear = (rear+1);
q[rear] = item;
count++;
}
int deletefront();
{
if (front > rear)
{
front=0;
rear = -1;
return -1;
}
return q[front++];
}
void displayq()
{
int i;
if (front > rear)
```



Lab-5

4) Circular Queue
a) Insert b) delete c) display

Page No. Date 19/10/2022

```
#include <stdio.h>
#include <stdlib.h>
#define que_SIZE 3
int item, front=0, rear=-1, q[que_SIZE], count=0;
void insertrear()
{
if(count==que_SIZE)
{
printf("queue overflow");
return;
}
rear=(rear+1)%que_SIZE;
q[rear]=item;
count++;
}
int deletefront()
{
if(count==0) return -1;
item=q[front];
front=(front+1)%que_SIZE;
count=count-1;
return item;
}
void display()
{
int i,f;
if(count==0)
{
```

```
printf ("queue is empty");
return i
}
f= front i
printf ("contents of queue \n");
for (i=0 ; i<= count ; i++)
{
printf ("%d \n", q[f]) ;
f= (f+1) % que.size;
}
}
Void main()
{
int choice ;
for (;;)
{
printf ("\n1. Insert rear \n 2. Delete front \n
        3. Display \n4. exit \n");
printf (" enter the choice");
Scanf ("%d \n", &choice);
switch (choice)
{
case 1 ; printf ("Enter the item to be inserted")
        scanf ("%d", &item);
        insertrear();
        break ;
case 2 : item = deletefront();
        if (item == -1)
        printf ("queue is empty \n");
```

```
else
printf("item deleted is %d\n", item);
break;
case 3: display();
        break;
default: exit(0);
    }
}
```

output:

1. insertrear
2. deletefront
3. display
4. exit

Enter the choice : 1
Enter the element to be inserted : 10
Enter the choice : 1
Enter the element to be inserted : 20
Enter the choice : 3
Contents of queue
10
20

```
1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 1
Enter the item to be inserted :10

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 1
Enter the item to be inserted :10

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 1
Enter the item to be inserted :30

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice : 1□ □3
contents of queue
10
10
30
10

1.Insert rear
2.Delete front
3.Display
4.exit
 Enter the choice :
```

5) Linked list :- inserting, deleting at rear & front end sorting, concat, display.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int info;
struct node *link;
};
typedef struct node *Node;
node getnode()
{
Node x;
x = (Node) malloc (sizeof (struct node));
if (x == NULL)
{
printf (" mem full \n");
exit (0);
}
return x;
}
void freenode (Node x)
{
free (x);
}
Node insert_front (Node first, int item)
{
Node temp;
temp = getnode();
```

```
temp->info=item;
temp->link=NULL;
if(first==NULL)
return temp;
temp->link=first;
first=temp;
return first;
}
node delete_front(Node first)
{
Node temp;
if(first==NULL)
{
printf("list empty cant delete\n");
return first;
}
temp=first;
temp=temp->link;
printf("item deleted at front end is=%d\n",
        front->info);
free(first);
return temp;
}
Node insert_rear(Node first, int item)
{
Node temp, cur;
temp=getnode();
temp->info=item;
temp->link=NULL;
```

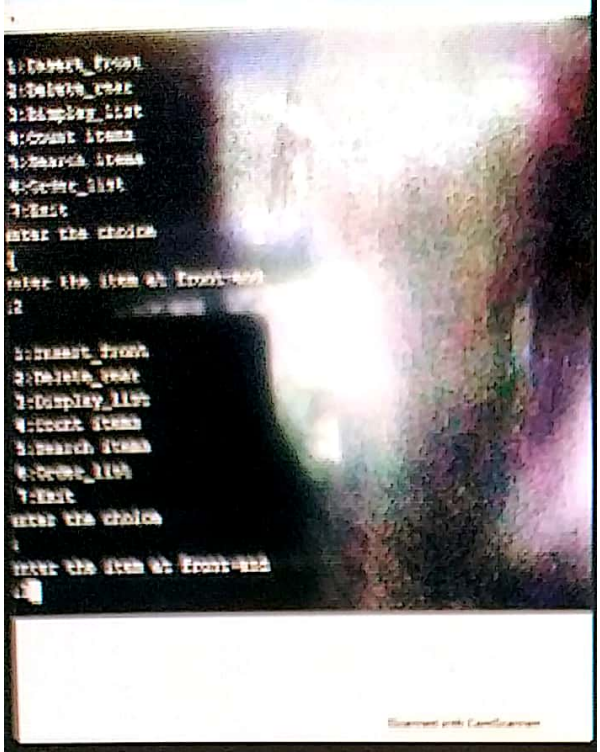
```
if(first == NULL);
return temp;
cur = first;
while(cur->link != NULL)
cur = cur->link;
cur->link = temp;
return first;
}
Node deleterear(Node first)
{
Node cur, prev;
if(first == NULL)
{
printf("list empty")
return first;
}
if(first->link != NULL)
{
prev = cur;
cur = cur->link;
}
printf("item deleted at rear-end is %d",
cur->info);
free(cur);
prev->link = NULL;
return first;
}
Node order_list(int item, node first)
{
```

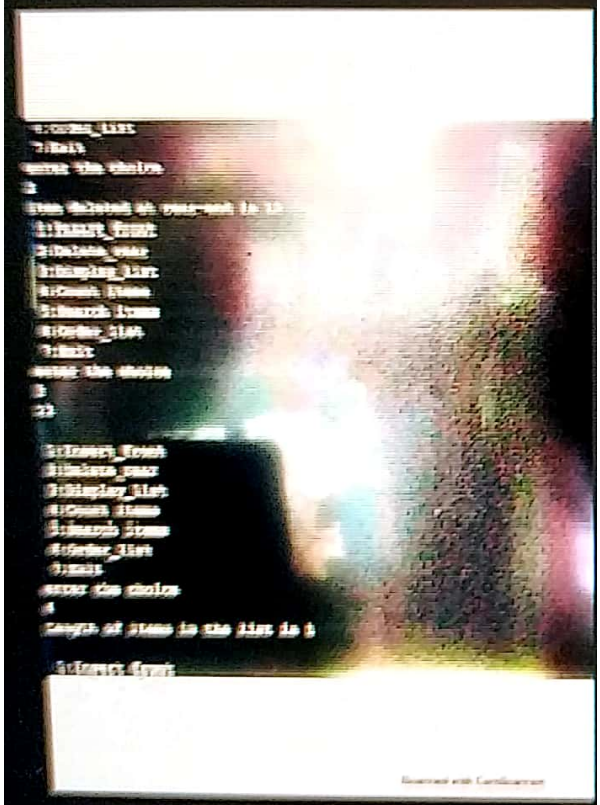
```
node temp,prev,cur;
temp=getnode();
temp->info=item;
temp->info=NULL;
if(first==NULL) return temp;
if(item<first->info)
{
temp->link=first;
return temp;
}
prev=NULL;
cur=first;
while(cur!=NULL && item>cur->info)
{
prev=cur;
cur=cur->link;
}
prev->link=temp;
temp->link=cur;
return first;
}
node delete_info(int key, node first)
{
node prev,cur;
if(first==NULL)
{
printf("list is empty\n");
return NULL;
}
```

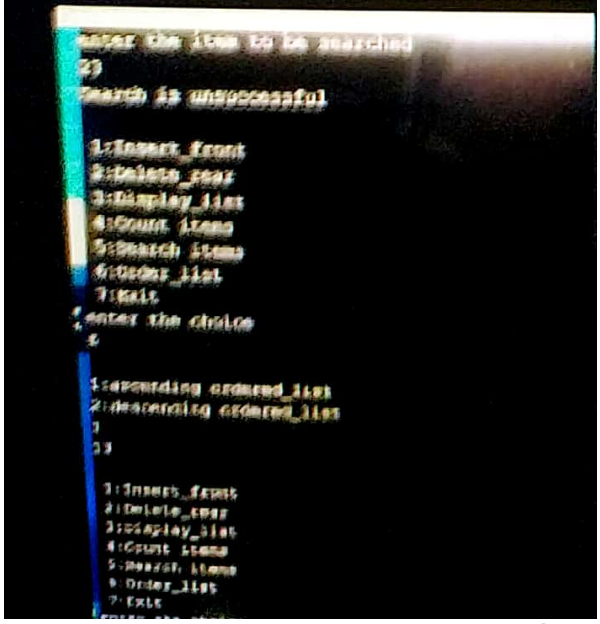
```
if(key = first->info)
{
cur=first;
first= first->link;
 freenode(cur);
return first;
}
prev= NULL;
cur = first;
 while (cur != NULL)
{
if (key == cur->info) break;
prev=cur;
cur = cur->link;
}
if (cur == NULL)
{
Print ("search is succesfull\n");
return first;
}
prev ->link == cur->link;
printf ("key deleted %d", cur->info);
freenode(cur);
return first;
}
Void display (Node first)
{
node temp;
if (first = NULL)
```

```
printf ("list empty");
for (temp=first ;temp!=NULL ; temp=temp->link)
{
printf ("%d\n", temp->info);
}
}
void main ()
{
int item, choice, key;
Node first=NULL;
for(;;)
{
printf ("1: insert_front\n 2: delete_front\n 3:
insert_rear\n 4: delete_rear\n"5: order_list\n 6:
delete_info\n 7: display\n 8: exit\n");
printf ("enter choice\n");
switch (choice)
{
case 1: printf ("enter the item at front end\n");
scanf ("%d", &item);
first= insert_front (first, item);
break;
case 2: first_delete_front (first);
break;
case 3: printf ("enter the item at rear_end\n");
scanf ("%d", &item);
first = insert_rear (first, item);
break;
```

```
case 4 : first = delete_rear(first);
    break;
case 5 : printf("enter the item to be inserted
                 list\n");
    scanf("%d", &item);
    first = order_list(item, first);
break;
case 6 : printf("enter the item to be delete
    scanf("%d", &key);
    first = delete_info(key, first);
    break;
case 7 : display(first);
    break;
default : exit(0);
break;
}
}
```







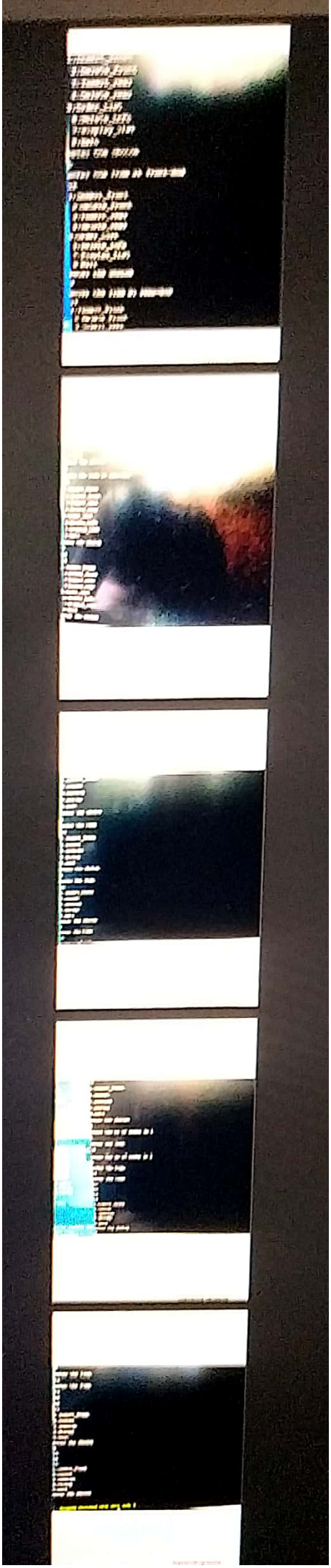
6) linked list: concating & reversing

```
#include <stdio.h>
#include <conio.h>
struct node
{
int info;
struct node * link;
};
typedf struct node * Node;
node getnode()
{
node x;
x= (node) malloc (sizeof (struct node));
if (x==NULL)
{
printf ("mem full");
exit(0);
}
return x;
}
Node insertrear (Node first, int item)
{
Node temp, cur;
temp=getnode();
temp->info = item;
temp->link = NULL;
if (first==NULL)
return temp;
cur=first;
```

```
while(cur->link != NULL)
cur = cur->link;
cur->link = temp;
return first;
}
void display (node first)
{
node temp;
if (first == NULL)
printf ("list empty\n");
for(temp= first; temp != NULL; temp= temp->link)
{
printf ("%d\n", temp->info);
}
}
node concat (node first, node second)
{
node cur;
if (first == NULL)
return second;
if (second == NULL)
return first;
cur = first;
while (cur->link != NULL)
cur = cur->link;
cur->link = second;
return first;
}
```

```
node reverse (node first)
{
node cur, temp;
cur = NULL;
while (first != NULL)
{
temp = first;
first = first -> link;
temp -> link = cur;
cur = temp;
}
return cur;
}
void main ()
{
int item, choice, pos, i, n;
node first = NULL, a, b;
for (;;)
{
printf ("1. insert front \n 2. concat \n 3. reverse \n
4. display \n 5. exit \n");
printf ("enter the choice \n");
scanf ("%d", &choice);
switch (choice)
{
case 1 : printf ("enter the item \n");
scanf ("%d", &item);
first = insert_rear (first, item);
break;
```

```
case 2: printf("enter no of nodes in 1\n");
        scanf("%d",&n);
        a= NULL;
        for(i=0;i<n;i++)
        {
        printf("enter the item\n");
        scanf("%d",&item);
        a=insert_rear(a,item);
        }
printf("enter no. of nodes in 2\n");
        scanf("%d",&n);
        b= NULL;
        for(i=0;i<n;i++)
        {
        printf("enter the item\n");
        scanf("%d",&item);
        b=insert_rear(b,item);
        }
a=concat(a,b);
display(a);
break;
case 3: first=reverse(first);
        display(first);
        break;
case 4: display(first);
        break
default exit(0);
}
}
```



7/12/2020

7) Impliment linked list: Insert front, display, sort, search, count

```
#include <stdio.h>
#include <stdlib>
struct node
{
int info;
struct node*link;
};
typedef struct node*NODE;
NODE getnode()
{
Node x;
x = (NODE)malloc(sizeof(struct node));
if(x==NULL)
{
printf("mem full\n");
exit(0)
}
return x;
}
Void freenode(NODE x)
{
free(x);
}
NODE insert_front(NODE first, int item)
{
NODE temp;
temp = getnode();
temp->info = item;
temp->link = NULL;
```

```
if (first == NULL)
  return temp;
temp -> link = first;
  first=temp;
return first;
}
NODE delete_rear (NODE first)
{
NODE cur, prev;
if (first == NULL)
{
printf (" list empty can't delete \n");
return first;
}
if (first -> link == NULL)
{
printf (" item deleted is %d\n", first->info);
free (first);
return NULL;
 }
prev = NULL;
cur = first;
while ( cur -> link != NULL)
{
prev = cur;
cur = cur -> link;
}
printf (" item deleted at rear-end is %d", cur->info);
free (cur);
```

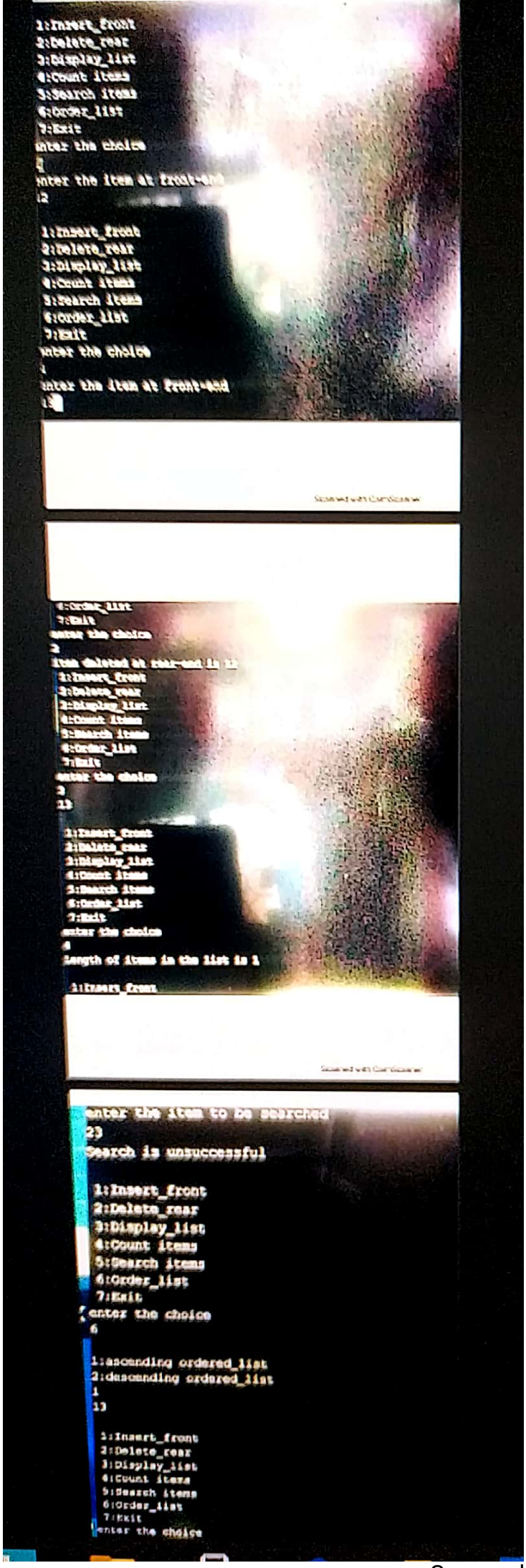
```
prev->link = NULL;
return first;
}
void display (NODE first
{
NODE temp;
if (first == NULL)
printf ("list empty can't display\n");
for(temp = first; temp != NULL; temp = temp->link)
{
printf("%d\n", temp->info);
}
}
int length(NODE first)
{
NODE cur;
int count = 0;
if (first == NULL)
return 0;
cur = first;
while (cur != NULL)
{
count c++;
cur = cur->link;
{
return count;
}
void search (int key, NODE first)
{
```

```
NODE cur;
if(first == NULL)
{
printf("list empty\n");
return;
}
cur = first;
while(cur != NULL)
{
if(key == cur->info)
break;
cur = cur->link;
}
if(cur == NULL)
{
printf("search unsuccessful\n");
return;
}
printf("search successfull\n");
}
NODE asc(NODE first)
{
NODE prev=first;
NODE cur = NULL;
int temp;
if(first == NULL{
return 0; }
else {
while(prev != NULL) {
```

```
cur = prev->link;
while (cur != NULL) {
if(prev->info > cur->info) {
temp = prev->info
prev->info = cur->info
cur->info = temp;
}
cur = cur->link;
}
prev = prev->link;
  }
}
return first;
}
NODE del (NODE first).
{
NODE prev = first;
NODE cur = NULL;
 int temp;
 if (first == NULL)
{
return 0;
}
else {
while (prev != NULL){
cur = prev->link;
while (cur != NULL) {
if (prev->info< cur->info) {
temp = prev->info;
```

```
prev->info = cur->info;
cur->info = temp;
}
cur = cur->link; }
prev = prev->link; }
return first;
}
int main() {
int item, choice, count, key, option;
NODE first = NULL;
for(;;) {
printf("\n1. Insert_front\n 2: Delete_rear\n 3: displa
_list\n 4:count_items\n 5: search items\n 6: ordu
list\n 7: exit\n");
printf("enter the choice\n");
scanf("%d", &choice);
switch(choice) {
case 1: printf("enter the item at front_end\n");
        scanf("%d", &item);
        first = insert_front(first, item);
        break;
case 2: first = delete_rear(first); break;
case 3: display(first); break;
case 4: count = length(first);
        printf("length = %d\n", count); break;
case 5: printf("enter the item to be searched\n");
        scanf("%d", &key);
        search(key, first);
        break;
```

```
case 6: printf ("\n1: ascending ordered list\n
2: descending Ordered list\n");
scanf ("%d", &option);
if (option == 1)
{
first = asc (first);
display (first);
}
else {
first = des (first);
display (first);
} break;
default : exit (0);
}
return 0;
}
```



8) program to implement stack using

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct node
{
int data
struct node* next;
};
Struct node * top = NULL;
  void push()
  {
  struct node * new_node;
  new_node = (struct node*) malloc
  printf (" enter the element\n");
  Scanf (" %d", & new_node->data);
    new node -> next = NULL;
  if (top == NULL)
  {
  top = new_node;
  }
  else
  {
```

top →

Scanned with CamScanner

1BM19CS166

```
new node->next = top;
top = new node;
}
}
void pop()
{
if (top == NULL)
printf ("stack empty\n");
else
{
temp = top;
while (temp != NULL)
{
printf ("%d\n", temp->data);
temp = temp->next;
}
}
}
void main()
{
int choice;
while(1)
{
printf ("1.push\n");
printf ("2.pop\n");
printf ("3.display\n");
printf ("enter choice");
scanf ("%d", &choice);
switch choice)
{
case 1: push(); break;
case 2: pop(); break;
case 3: display(); break;
default: exit(0);
}
}
}
```

```
 4:Exit
enter the choice
1
enter the item at front-end
12

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=12

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
12

 1:Insert_front
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
```

// Implementing queue using linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int node
{
struct node* next)
{
struct node* front=NULL, *rear=NULL;
void insert()
{
struct node * new_node)
new_node=(struct node*)malloc(sizeof(struct node)
printf("enter element\n");
scanf("%d", &new_node->data);
new_node->next=NULL;
if(rear==NULL)
{
rear==NULL)
{
rear=new_node;
front=new_node;
}
else
{
rear->next=new_node;
rear=new_node;
}
}
void del()
{
if(front==NULL)
printf("queue is empty\n");
else
printf("deleted element is %d", front->data);
```

Scanned with CamScanner

```
front=front->next;
}
}
void display()
{
struct node *temp;
if (front == NULL)
printf ("queue is empty\n");
temp = front;
while (temp != NULL)
{
printf ("%d\n", temp->data);
temp=temp->next;
}
}
void main()
{
int choice;
while(1)
{
printf ("1. insert\n2. delete\n3. display\n4. exit\n");
scanf ("%d", &choice);
switch(choice)
{
case 1: enqueue(); break;
case 2: del(); break;
case 3: display(); break;
case 4: exit(0);
}
}
```

```
 4:Exit
enter the choice
1
enter the item at rear-end
12

1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
2
item deleted at front-end is=12

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
3
list empty cannot display items

 1:Insert_rear
 2:Delete_front
 3:Display_list
 4:Exit
enter the choice
```

Lab-program Doubly Linked List 14/12/2021

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int info;
struct node* llink;
struct node* rlink;
};
typedf struct node *NODE;
NODE getnode()
{
NODE x;
x=(NODE) malloc (sizeof(struct node));
if(x==NULL)
{
printf("memory full");
exit(0);
}
return x;
}
Void freenode(NODE x)
{
free(x);
}
NODE dinsert_front(int item, NODE head)
{
NODE temp, cur;
temp=getnode();
```

```
temp->info = item;
cur = head->rlink;
head->rlink = temp;
temp->llink = head;
temp->rlink = cur;
cur->llink = temp;
return head;
}
NODE dinsert_rear (int item, NODE head)
{
NODE temp, cur;
temp = getnode();
temp->info = item;
cur = head->llink;
head->llink = temp;
temp->rlink = head;
temp->llink = cur;
cur->rlink = temp;
return head;
}
NODE ddelete_rear (NODE head)
{
NODE cur, prev;
if (head->rlink == head)
{
printf ("empty\n");
return head;
}
```

```
    cur = head->link;
    prev = cur->link;
    head->link = prev
    prev->rlink = head;
    printf("The node deleted is %d", cur->info);
    freenode(cur);
    return head;
}
NODE insert_leftpos(int item, NODE head)
{NODE temp, cur, prev;
if (head->rlink == head)
{
printf("list empty\n");
 return head;
}
 cur = head->rlink;
while (cur != head)
{
if (item == cur->info) break;
 cur = cur->rlink;
}
if (cur == head)
{
printf("key not found");
 return head;
}
prev = cur->llink;
printf("enter toward left of %d = ", item);
```

```
temp = getnode();
scanf("%d", &temp->info);
prev->rlink = temp;
temp->llink = prev;
cur->llink = temp;
temp->rlink = cur;
return head;
}
NODE insert_rightpos(int item, NODE head)
{
NODE temp, cur, next;
if(head->rlink == head)
{
printf("list empty\n");
return head;
}
cur = head->rlink;
while(cur != head)
{
if(item == cur->info) break;
cur = cur->rlink;
}
if(cur == head)
{
printf("key not found\n");
return head;
}
next = cur->rlink;
```

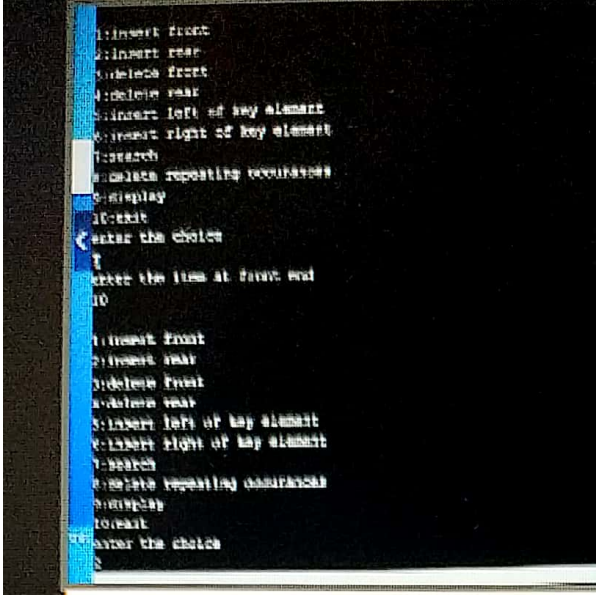
```
printf("enter towards right of %d =", item);
temp = getnode;
scanf("%d", &temp->info);
cur->rlink = temp;
temp->llink = cur;
next->llink = temp;
temp->rlink = next;
return head;
}
NODE search(NODE head, int item)
{
NODE temp, cur;
int flag=0;
if (head->rlink==head)
{
printf("list empty\n");
return head;
}
cur = head->rlink;
while (cur != head)
{
if (item == cur->info)
{
flag=1;
break;
}
cur = cur->rlink;
}
```

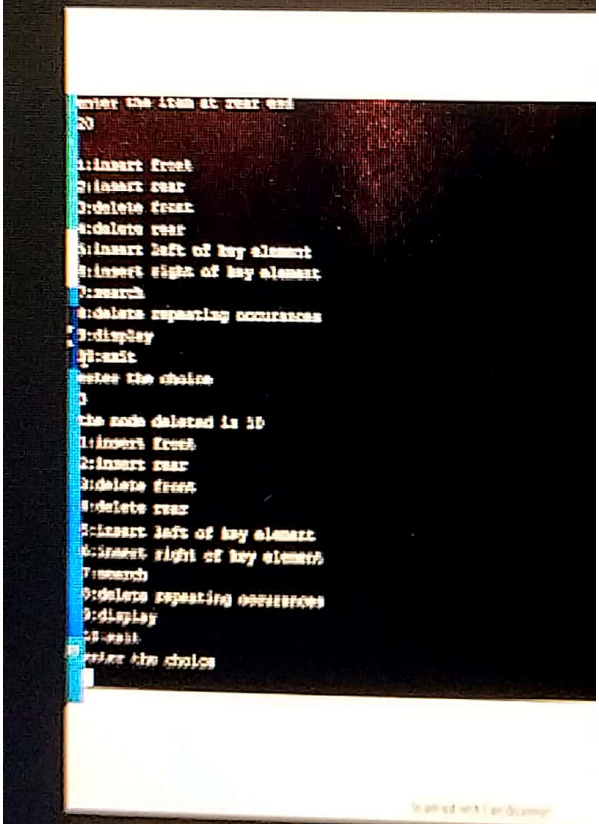
```
if (cur==head)
printf ("search unsuccessfull\n");
  if (flag==1)
  printf ("search successfull\n");
}
NODE delete_all_key (int item, NODE head)
{
NODE prev, cur, next;
int count;
  if (head->rlink==head)
{
printf("list empty\n");
return head;
}
count = 0;
cur = head->rlink;
while (cur != head)
{
if (item != cur->info)
cur = cur->rlink;
  else
{
count++;
prev = cur->link;
next = cur->rlink;
prev->rlink = next;
next->llink = prev;
  freenode(cur);
  cur = next;
}
```

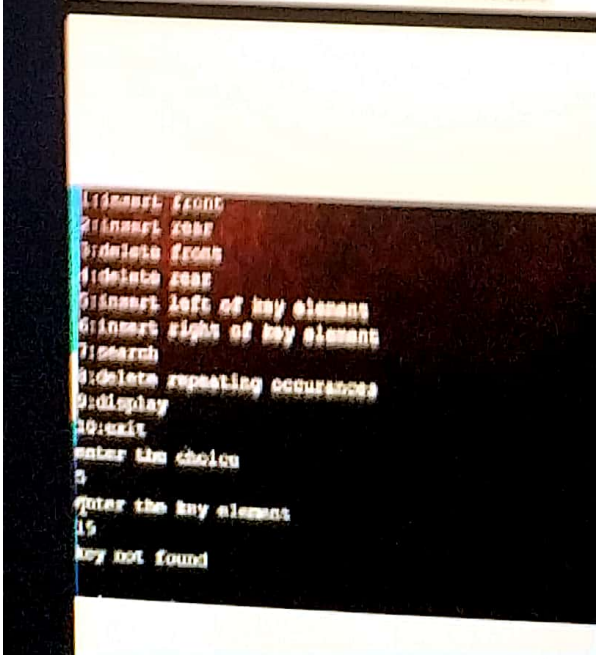
```
    if (count == 0)
    printf ("not found\n");
      else
    printf ("found at %d position & deleted", count);
        return head;
    }
}
Void display (NODE head)
{
NODE temp;
  if (head->rlink == head)
  {
  printf ("dq is empty\n");
  return;
  }
  printf ("contents of dq\n");
  temp = head->rlink;
while (temp != head)
{
printf ("%d", temp->info);
temp = temp->rlink;
}
printf ("\n");
}
void main()
{
NODE head, last;
int item, choice;
```

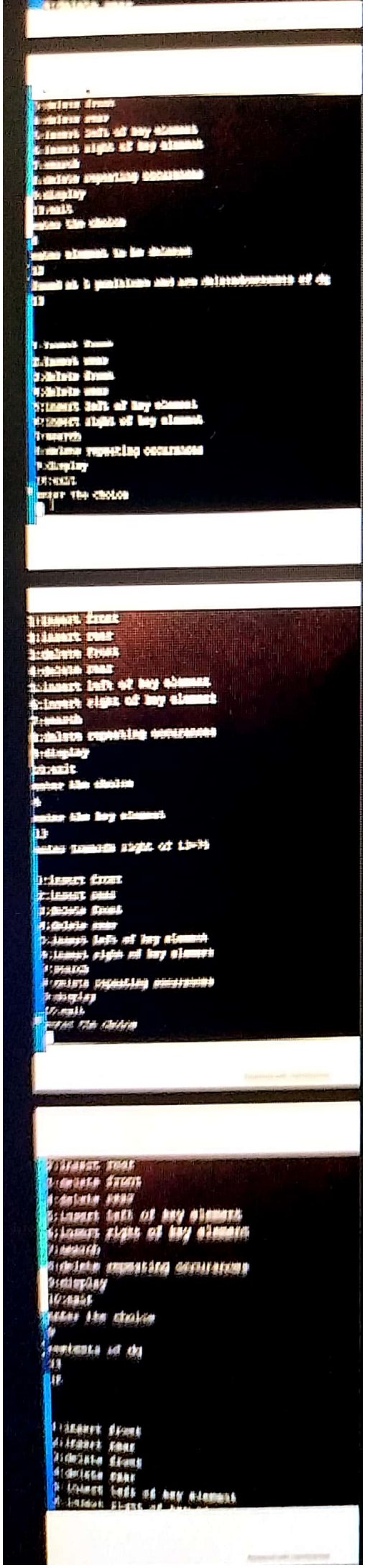
```
head = getnode()
head->rlink = head;
head->llink = head;
for(;;)
{
printf("\n1: insert front\n2: insert rear\n3: delete front\n4:
delete rear\n5: insert left of key element\n6:
insert right of key element\n7: search\n8: delete
repeating element\n9: display\n10: exit\n");
printf("enter the choice\n");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("enter the item at front end\n");
scanf("%d", &item);
last = dinsert_front(item, head); break;
case 2: printf("enter the item at rear end\n");
scanf("%d", &item)
last = dinsert_rear(item, head);
break;
case 3; last = ddeletefront(head); break;
case 4; last = ddelete_rear(head); break;
case 5: printf("enter the key element\n");
scanf("%d", &item);
last = insert_leftpos(item, head); break;
case 6: printf("enter the key element\n");
scanf("%d", &item);
last = insert_rightpos(item, head);
break;
```

```
case 7: printf ("enter the search element \n");
        scanf ("%d", &item);
        Search (head, item);
        break;
case 8: printf ("enter the element to be deleted &s duplicates \n");
        scanf ("%d", &item);
        last = delete_all_key (item, head);
case 9: display (head);
        break;
default: exit (0);
}
}
```









10. BINARY SEARCH TREE

```
#include <stdlib.h>
#include <stdio.h>
 struct node
 {
 int info;
 struct node *rlink;
 struct node *llink;
 };
 typedef struct node *NODE;
  NODE getnode()
  {
  NODE x;
  x = (NODE) malloc (sizeof (s
   if (x == NULL)
  {
  printf ("memory full\n");
   exit(0)
  }
  return x;
  }
  void freenode (NODE x)
  {
  free(x);
  }
  NODE insert (NODE root,
   {
   NODE temp, cur, prev;
   temp = getnode();
```

```
temp->rlink = NULL;
temp->llink = NULL;
temp->info = item;
prev = NU
return temp;
prev = NULL;
cur = root;
while (cur != NULL)
{
prev = cur
cur = (item < cur->info) ? cur->llink
}
if (item < prev->info)
prev->llink = temp;
else
prev->rlink = temp;
return root;
}
void display (NODE root, int i)
{
int j;
if (root != NULL)
{
display (root->rlink, i+1);
for (j=0; j<i; j++)
printf ("  ");
printf ("%d\n", root->info);
display (root->llink, i+1);
}
}
```

Root
pre :-
post :-
in :

```
Void preorder (NODE root)
{
if (root != NULL)
{
printf ("%d\n", root->info);
preorder(root->llink);
preorder (root->rlink);
}
}
void postorder (NODE root)
{
if(root != NULL)
{
postorder (root->llink);
postorder(root->rlink);
printf("%d\n", root->info);
}
}
void inorder (NODE root)
{
if (root != NULL)
{
inorder (root->llink);
printf ("%d\n", root->info)
inorder(root->rlink);
}
}
Void main ()
```

```
{
int item, choice;
NODE root = NULL;
for(;;)
{
printf("\n1.insert\n2.display\n3.pre\n4.post\n5.en 6e
printf("enter the choice\n");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("enter the item\n");
        scanf("%d", &item);
        root = insert(root, item)
        break;
case 2: display(root, 0)
        break;
case 3: preorder(root);
        break;
case 4; postorder(root);
        break;
case 5: inorder(root);
        break;
default: exit(0);
        break;
}
}
}
```

searching