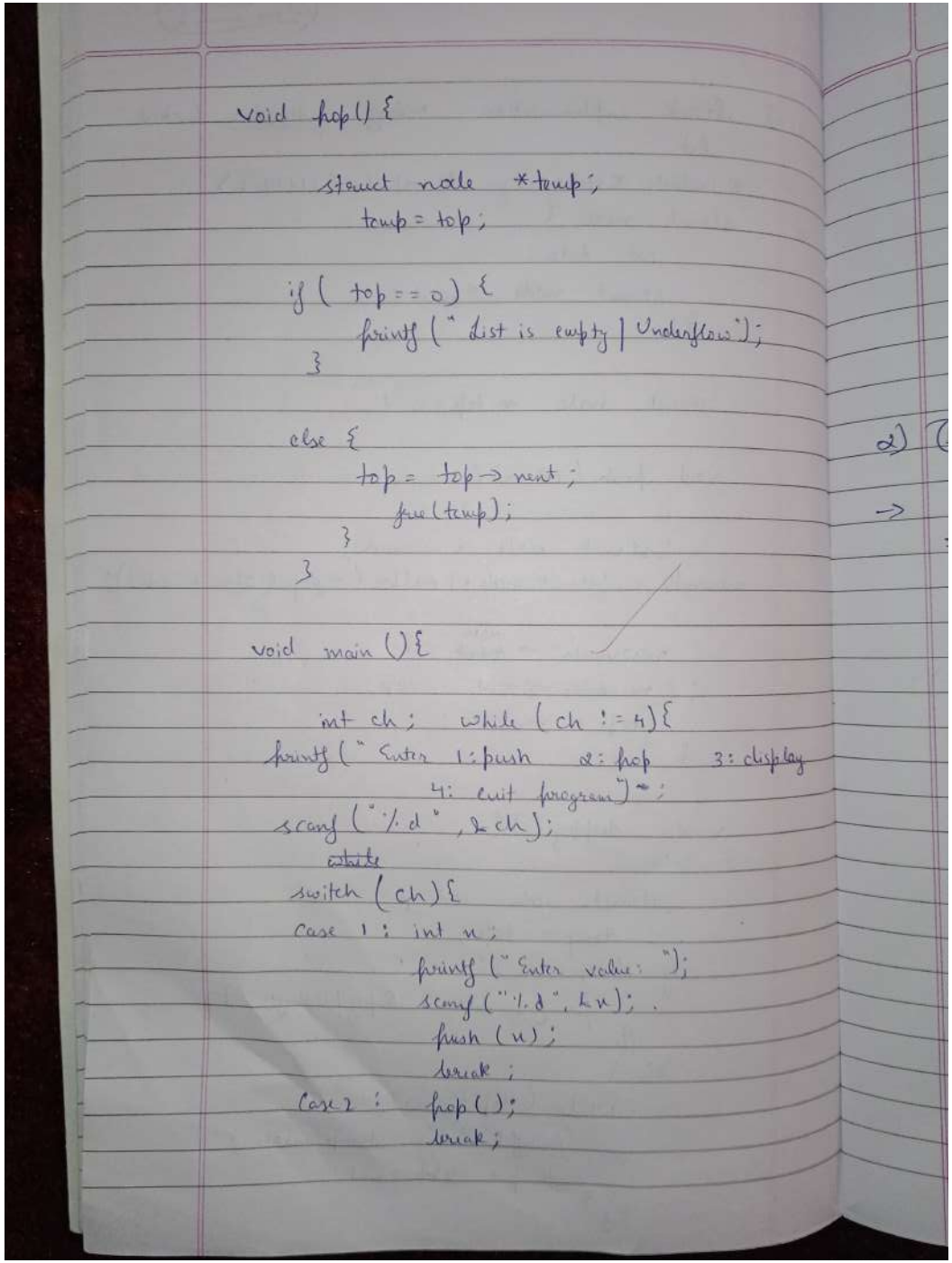
29/1/24

a) ⓐ Stack implementation using single linked list.

→ #include <stdio.h> #include <stdlib.h>

```
struct node {
    int data;
    struct node *next;
};

struct node * top = 0 ;

void push ( int n ) {

    struct node * newnode;
newnode = (struct node *) malloc (sizeof (struct node));

    newnode -> ~~next~~ data = n;
    newnode -> next = top;
    top = newnode;
}

void display ( ) {

    struct node *temp;
        temp = top ;
    if ( top == 0 ) {
        printf (" List is Empty / Underflow ");
    }
    else {
        while ( top != 0 ) {
            printf (" %d ", temp -> data );
            temp = temp -> next;
    }
```

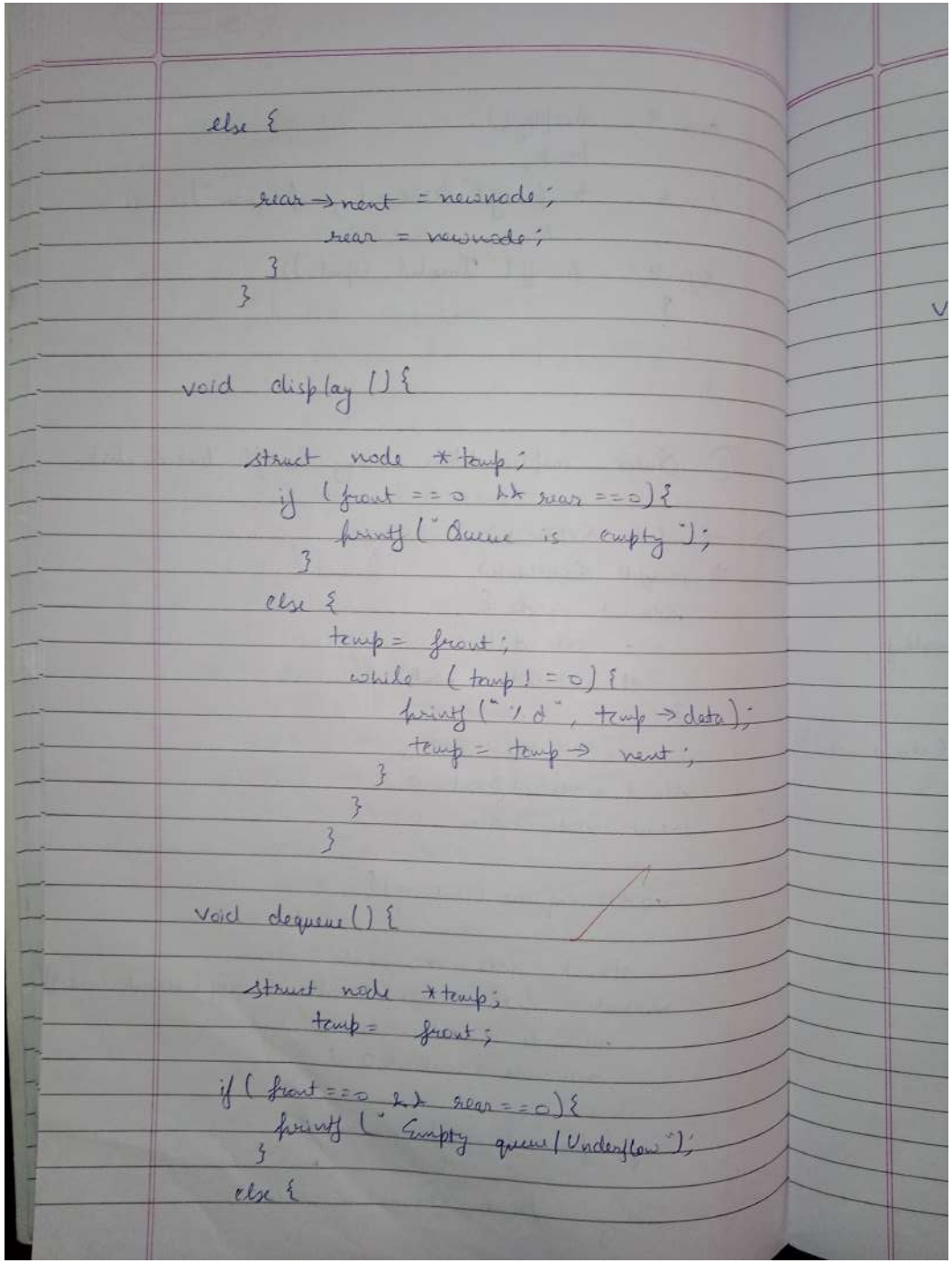


```
void pop() {

    struct node *temp;
      temp = top;

  if ( top == 0) {
      printf (" list is empty | Underflow");
   }

  else {
      top = top -> next;
        free(temp);
    }
 }

void main (){

    int ch;  while ( ch != 4){
printf (" Enter 1:push  2: pop  3: display
          4: exit program");
scanf ("%d", &ch);
  switch
 switch (ch){
 case 1 : int n;
          printf ("Enter value: ");
          scanf ("%d", &n);
           push (n);
           break ;
 case 2 :  pop ();
           break ;
```

```
case 3: display();
        break;
case 4: printf("terminating program");
        break;
default: printf("Invalid input");
    }
  }
}
```

a) (b) Queue implementation using single linked list.

→
```
#include <stdio.h>
#include <stdlib.h>
  struct node {
        int data;
    struct node * next;
};

struct node *front = 0;
struct node *rear = 0;

void enqueue (int x){

  struct node *newnode;
newnode = (struct node *) malloc (sizeof (struct node));
   newnode → data = x;
   newnode → next = 0;

if (front == 0 && rear == 0){

        front = rear = newnode;
}
```



```
else {
    rear -> next = newnode;
        rear = newnode;
    }
}

void display () {
    struct node *temp;
    if (front == 0 && rear == 0) {
        printf ("Queue is empty");
    }
    else {
        temp = front;
        while (temp != 0) {
            printf ("%d", temp -> data);
            temp = temp -> next;
        }
    }
}

void dequeue () {
    struct node *temp;
        temp = front;
    if (front == 0 && rear == 0) {
        printf ("Empty queue / Underflow");
    }
    else {
```

```
        front = front -> next;
            free (temp);
        }
    }

Void main ( ) {
    int ch;
while (ch != 0) {
printf (" Enter :- 1 : enqueue  2 : dequeue
        3 : display    0 : exit program ");
scanf ("%d", &ch);

switch(ch) {

case 1 : int x ;
        printf (" Enter value : ");
        scanf ("%d", &x);
        enqueue (x);
        break;
case 2 : dequeue ();
        break;
case 3 : displayQ;
        break;
case 0 : printf (" terminating program ");
        break;
default : printf (" Invalid input ");
        break;
  }
 }
}
```

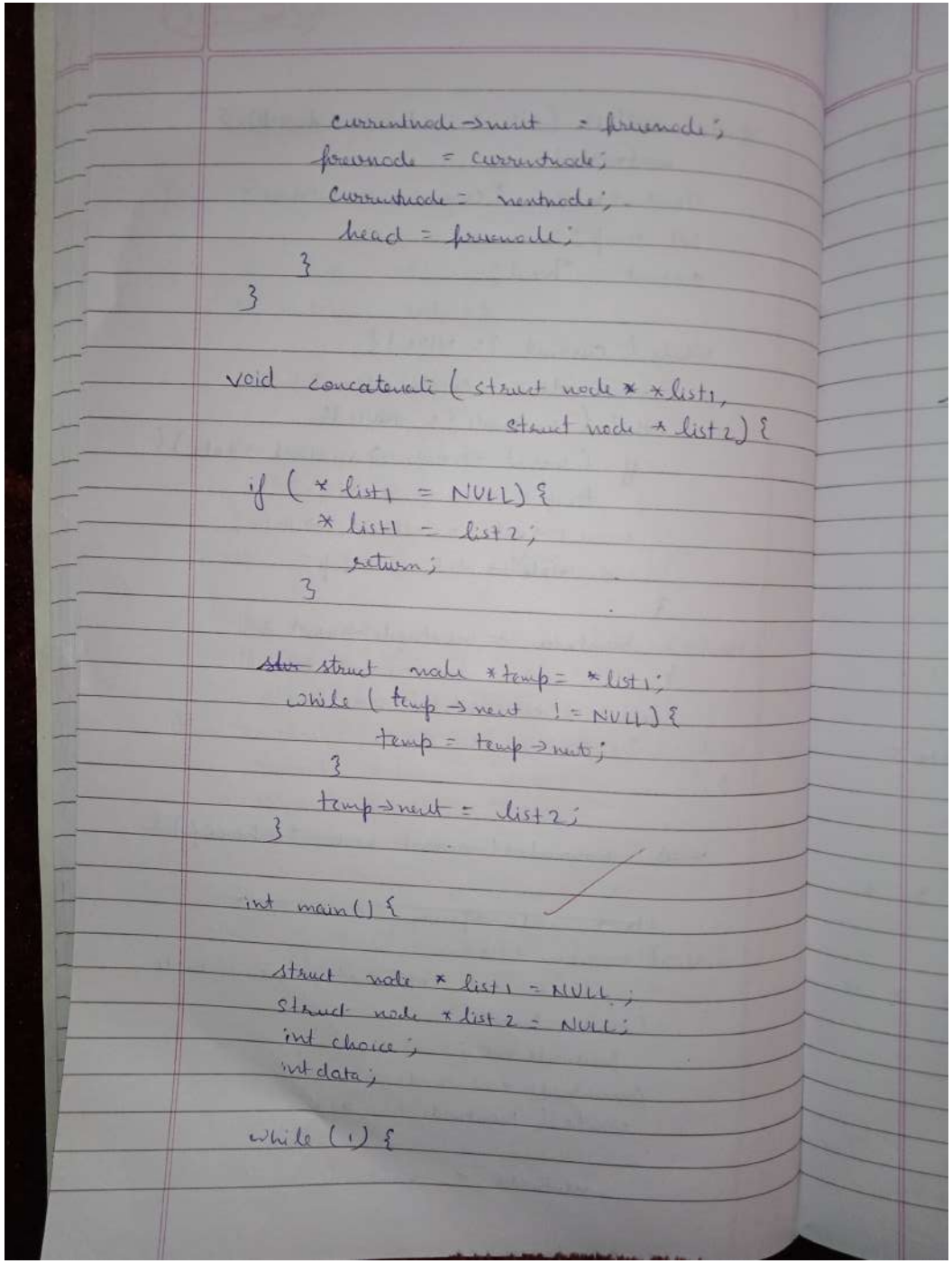
Output

1) Sorting, reversing and concatenation

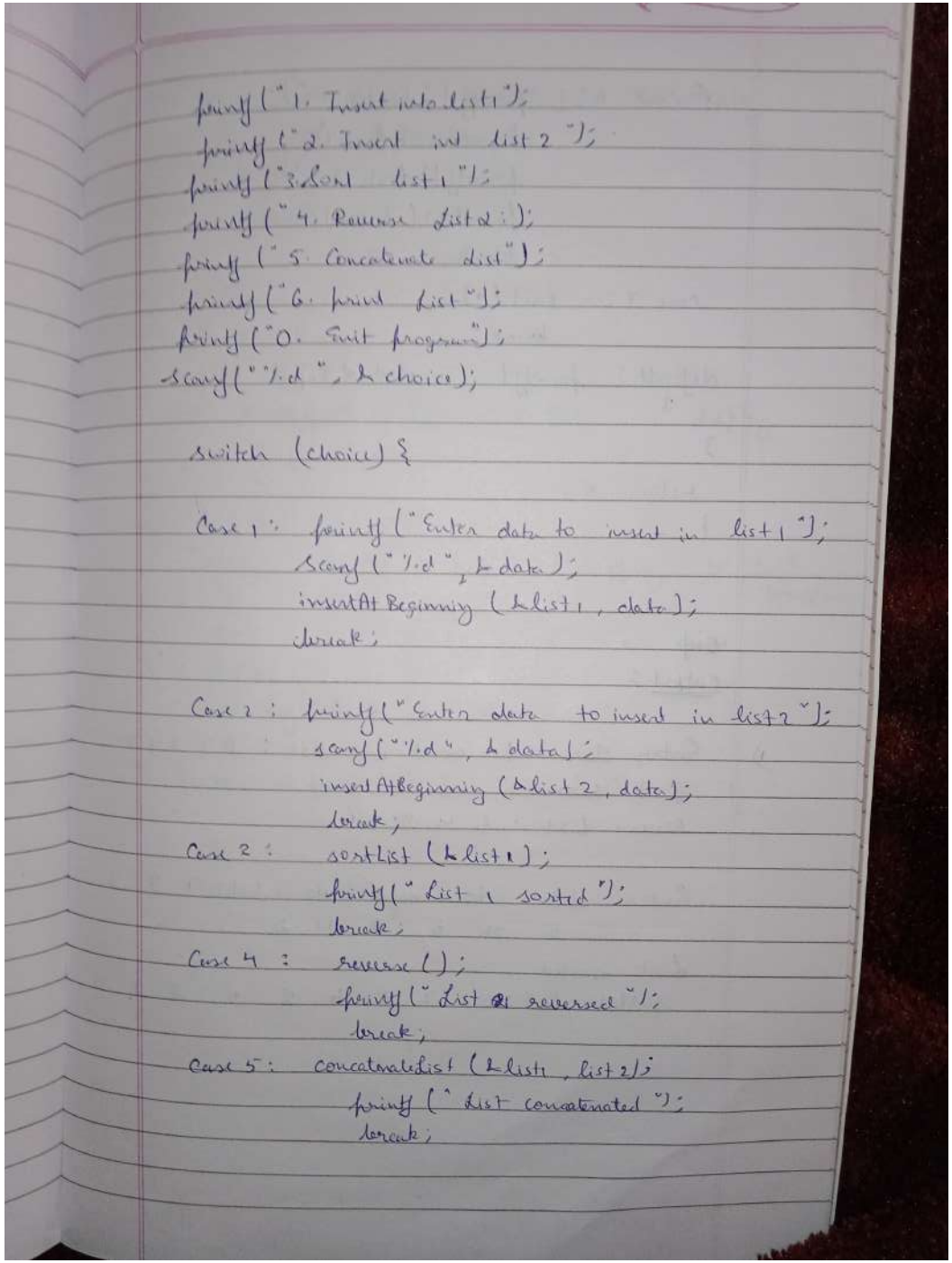
→

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

struct node {
    int data;
    struct node * next;
};

void sortList (struct node **head){

void insertAtBeginning ( struct node **headRef,
    int data) {
  struct node *newnode
  struct node *newNode = ;
newnode = (struct node *) malloc (sizeof (struct node
  newnode -> data = data;
  newnode -> next = *headRef;
  *headRef = newnode;
}

void printList( struct node *head){
      while ( head != NULL){
      printf(" %d", head -> data);
      head = head -> next;
  }
```

```
void sortlist (struct node **head) {
    nextnode
    struct node *current, *nextnode;
    int temp;
    current = *head;

    while ( current != NULL ){
        nextnode = current -> next;
        while ( nextnode != NULL){
            if (current -> data > nextnod -> data ) {
                temp = current -> data;
            current -> data = nextnode -> data;
            nextdata node -> data = temp;
        }
            nextnode = nextnode -> next;
        }
        current = current -> next;
    }
}


void reverselist( struct node **headref) {

    struct node *prevnode, *current, *
void reverse () {
    struct node *prevnode, *currentnode,
    *nextnode;
    prevnode = 0;
    currentnode = nextnode = head;
    while ( nextnode != NULL) {

        nextnode = nextnode -> next;
```



```
        currentnode->next = frwnode;
        frwnode = currentnode;
        currentnode = nextnode;
        head = frwnode;
    }
}

void concatenate (struct node **list1,
                  struct node *list2) {

if (*list1 == NULL) {
    *list1 = list2;
    return;
}

struct node *temp = *list1;
while (temp->next != NULL) {
    temp = temp->next;
}
temp->next = list2;
}

int main() {

struct node *list1 = NULL;
struct node *list2 = NULL;
int choice;
int data;

while (1) {
```



```
printf("1. Insert into list1");
printf("2. Insert in list 2");
printf("3. Sort list 1");
printf("4. Reverse List 2:");
printf("5. Concatenate list");
printf("6. print list");
Printf("0. Exit program");
scanf("%d", &choice);

switch (choice) {

Case 1: printf("Enter data to insert in list1");
        scanf("%d", &data);
        insertAtBeginning(&list1, data);
        break;

Case 2: printf("Enter data to insert in list2");
        scanf("%d", &data);
        insertAtBeginning(&list2, data);
        break;
Case 3: sortList(&list1);
        printf("List 1 sorted");
        break;
Case 4: reverse();
        printf("List reversed");
        break;
Case 5: concatenateList(&list1, list2);
        printf("list concatenated");
        break;
```

```
case 6: printf("List1: ");
        printList(list1);
        printf("List 2: ");
        printlist(list2);
        break;
case 7: init(0);
        break;
default: printf("Invalid input");
}
}
return 0;
}
```

~~Oip~~

output:

1) Enter data to insert into list: 1 2 3 4 5
List reversed
Reverse list: 5 4 3 2 1

Enter data to insert into list: 1 3 6
5 4 8 9 11 2
List sorted
sorted list: 1 2 3 4 5 6 8 9 11

9 (a) Enter 1: push 2: pop 3: display
4: exit program

~~push~~ 1
Enter value : 5
push(3)
push(2)

Enter ~~display~~ 1: push 2: pop 3: display
4: exit program
3
2 3 5
Enter 1: push 2: pop 3: display 4: exit program

2
pop()
display()
5
Enter 1: push 2: pop 3: display 4: exit program

4
terminating program

5/12/2024

2) (b) Enter 1: enqueue 2: dequeue 3: display
0: Exit program

1

Enter value: 6

enqueue(4)

enqueue(1)

Enter 1: enqueue 2: dequeue 3: display
0: exit program

3

6 4 1

dequeue()

dequeue()

display()

1

Enter 1: enqueue 2: dequeue 3: display
0: exit program

0

terminating program.