1. C Program for finding the 2nd largest element in an integer array

AIM:
To find the 2nd largest element in an integer array

Algorithm:
1. Initialize an array, its size, and variables to store the first and second largest elements.
2. Input the size of the array and its elements
3. Initialize the first and second largest elements as the smallest possible integer values.
4. Iterate through the array and update the first and second largest elements based on the current element.
The algorithm's time complexity is O(n) since it iterates through the array once to find the two largest elements, where n is the size of the array.

Coding:

```
#include <stdio.h>
#include <limits.h>
#define MAX_SIZE 1000
int main() {
int arr[MAX_SIZE], size, i;
int max1, max2;
Printf("Enter size of the array (1-1000): ");
Scanf("%d", &size);
Printf("Enter elements in the array: ");
for (i=0; i< size, i++) {
Scan ("%d", &arr[i]);
}
```

```
max1 = max2 = INT_MIN;

for(i=0; i<size; i++){
  if(arr[i] > max1){
    max2 = max1;
    max1 = arr[i];
  } else if (arr[i] > max2 && arr[i] < max1){
    max2 = arr[i];
  }
}
printf("first largest = %d\n", max1);
printf("second largest = %d", max2);
return 0;
```

Output:

```
Enter size of the array(1-1000): 5
Enter elements in the array: 45
76
90
89
70
Fist largest = 90
Second largest = 89
```

Result:

Thus the program is successfully implemented to find 2nd largest element in the given ar

2. C program to print only the odd numbers in odd indices of an given integer array.

AIM:
To find only the odd numbers in odd indices of an given integer array.

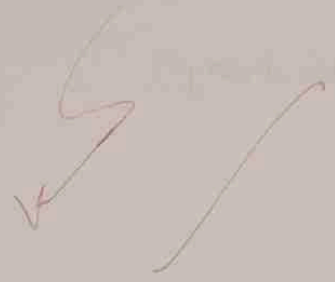
Algorithm:
1. Input the size of the array and its elements.
2. Iterate thorough the array at odd indices and check if the number at that index is odd.
3. If the number at the odd index is odd, then print it.

Coding:

```
#include<stdio.h>
#define MAX_SIZE 1000
int main(){
printf("Enter size of the array(1-1000):");
scanf("%d", &size);

printf("Enter elements in the array:");
for(i=0;i<size;i++){
  scanf("%d", &arr[i]);
}

printf("odd numbers in odd indices are:");
for(i=1;i<size,i+=2){
  if(arr[i]%2!=0){
   Printf("%d", arr[i]);
  }
}

return 0;
```

Output:

```
Enter size of the array (1-1000): 3
Enter elements in the array: 2

3
4
odd numbers in odd indices are: 3
```



Result:

Thus the program is successfully implemented find only odd numbers in odd indices of an en integer array.

3. write a C program for Linear Search

AIM:
C program for Linear Search

Algorithm:
1. Initialize an array, its size, and variables to store the search number and location.
2. Input the size of the array and its elements.
3. Iterate through the array and store the elements in the array.
4. Input the number to search for.
5. Iterate through the array and compare the search number with each element.
6. If the search number is found, print its location and exit the loop.
7. If the search number is not found, print a message indicating that the number is not present in the array.

Coding:

```
#include <Stdio.h>

int main(){
 int array[100], search, c, n;
printf("Enter number of elements in array\n");
scanf("%d", &n);

printf("Enter %d integer(s)\n", n);

for(c=0; c<n; c++)
scanf("%d", &array[c]);
Printf["Enter a number to search\n");
Scanf("%d", & Search);
for(c=0; c<n; c++){
if (array[c] == search){
```

```
printf("%d is present at location %d.\n", search, c+1);
break
}
}
if (c == n)
printf("%d isn't present in the array.\n", search);

return 0;
}
```

output:

```
/tmp/lxJ35hGhpl.o
Enter number of elements in array
5
Enter 5 integer(s)
3
5
8
4
9
Enter a number to search
8
8 is present at location 3.
```

Result:

Thus the program is successfully implemented to find linear search.

4. write a C program for binary search.

AIM:
C program for binary search.

Algorithm:
1. Define a function called binary search that takes an array, its size, and a key value as input parameters.
2. Initialize three variables: low, high, and mid. set low to 0 and hight to the size of the array minus 1.
3. while low is less than or equal to high; calculate the middle index using the formula mid = (low + high) / 2
4. Check if the element at the middle index is equal to the search key. If so, return the middle index.
5. If the element at the middle index is less than the search key, update low to be the middle index plus 1. otherwise, update high to be the middle index minus 1.
6. Repeat steps 3 to 5 until low is greater than high.
7. If the search key is not found, return -1

Coding:

```
#include<stdio.h>

int binary search (int arr[], int n, int key){
 int low = 0;
 int high = n-1;

while (low <= high){
 int mid = (low+high)/2;

if(arr[mid] == key){
 return mid;
} else if (arr[mid] < key){
```

```
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}
return -1;
}
int main() {
int n;
printf("Enter the size of the array: ");
scanf("%d", &n);
int arr[n];
printf("Enter the elements of the array: ");
for (int i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
int key;
printf("Enter the element to search: ");
scanf("%d", &key);
int result = binarySearch(arr, n, key);
if (result != -1) {
printf("Element found at index: %d", result);
} else {
printf("Element not found");
}
return 0;
}
```

Output:

```
Enter the size of the array: 5
Enter the elements of the array: 2
6
4
3
7
Enter the element to search: 7
Element found at index: 4
```

Result:

Thus the program is successfully implemented for Binary Search.

5. write a C program for linked list implementation of stack.

AIM:
C program for linked list implementation of stack

Algorithm:
1. Define a struct node that contains an integer and a pointer to another node.
2. Initialize a pointer top to NULL.
3. Define four functions: push, pop, and display.
   * push: Create a new node with the given data, set its next pointer to the current top, and update the top pointer to the new node.
   * pop: If the top pointer is NULL, return -1. Otherwise, retrieve the data from the top node, update the top pointer to the next node, and free the old top node.
   * display: print the data of each node in the stack, followed by a newline character.

Coding:

```
#include<stdio.h>
#include<stdlib.h>
Struct node{
int data;
Struct node *next;
};
Struct node *top=NULL;
Void push (int data){
Struct node *newNode=(Struct node*)malloc
                     (sizeof(Struct node));
```

```
newNode->data = data;
newNode->next = top;
top = newNode;
}
int pop(){
if(top == NULL){
return -1;
}
int data = top->data;
struct node *temp = top;
top = top->next;
free(temp);
return data;
}
void display(){
struct node *ptr = top;
while(ptr != NULL){
printf("%d,", ptr->data);
ptr = ptr->next;
}
printf("\n");
}
int main(){
push(1);
push(2);
push(3);
push(4);
printf("stack elements: ");
display();
printf("poped element: %d\n", pop());
printf("poped element: %d\n", pop());
```

```
printf("Stack elements: ");
display();
return 0;
}
```

Output:

```
Stack elements : 4 3 2 1
Poped element : 4
Poped element : 3
Stack elements : 2 1
```

Result:

Thus the program is successfully implemented to find the linked list implemention of stack.

6. write a C program for implementing infix to postfix conversion using Stack.

AIM:

C program for implementing Infix to postfix Conversion using stack.

Algorithm:

1. Define a char array stack and initialize a variable top to -1.
2. Define a function push that pushes a character into the stack.
3. Define a function pop that removes and returns the top character from the stack.
4. Define a function priority that takes a character and returns its priority in the expression (0 for parentheses, 1 for +/-, 2 for */, and 3 for /).
5. In the main function, read and expression from the user and iterate through the characters.
   * If the character is an operand, print it.
   * If the character is an operator, pop operators from the stack and print them until the stack is empty or the current operator has higher priority.
   * If the character is a closing parenthesis, pop all operators from the stack and print them.

Coding:

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

char stack[100];
int top = -1;
```

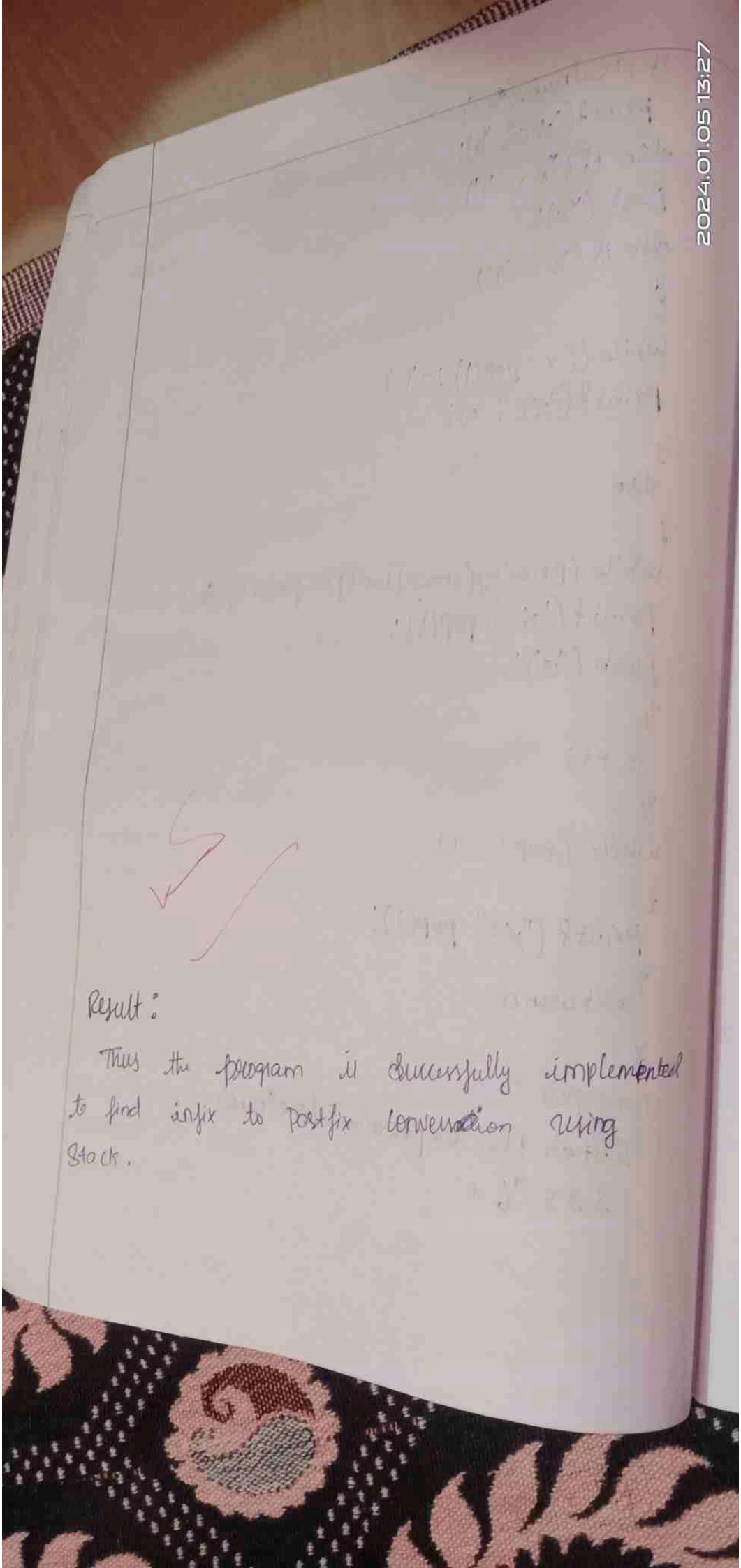
```
void push (char x)
{
stack[++top] = x;
}
char pop()
{
if(top == -1)
return -1;
else
return stack[top--];
}
int priority(char x)
{
if(x == '(')
return 0;
if(x == '+' || x == '-')
return 1;
if(x == '*' || x == '/')
return 2;
return 0;
}
int main()
{
char exp[100];
char *e, x;
printf("Enter the expression: ");
scanf("%s", exp);
printf("\n");
e = exp;
while(*e != '\0')
{
```

```
if (isalnum(*e))
 printf("%c", *e);
else if (*e == '(')
push (*e);
else if (*e == ')')
{
while ((x = pop()) != '(')
printf("%c", x);
}
else
{
while (priority(stack[top]) >= priority(*e))
printf("%c", pop());
push (*e);
}
 e++;
}
while (top != -1)
{
 printf ("%c", pop());
}return 0;
}
```

Output:

Enter the expression: (38*5+6)

385*6+



Result:

Thus the program is successfully implemented to find infix to postfix conversion using stack.

7. write a C program for array implementation of queue.

AIM:
C program for array implementation of queue.

Algorithm:
1. Define a queue as an array of a fixed size and initialize the front and rear indices to -1.
2. Implement the enqueue operation to add elements to the rear of the queue checking for queue overflow.
3. Implement the dequeue operation to remove elements from the front of the queue, Checking for queue underflow.
4. Implement the display operation to show the elements in the queue.
5. In the main function, use a continuous loop to prompt the user for their choice of operation and call the corresponding function.
6. Queue.

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

#define MAX_SIZE 5

int queue[MAX_SIZE];
int front = -1, rear = -1;
void enqueue(int data){
 if (rear == MAX_Size-1){
```

```
printf("Queue overflow\n");
return;
}
if (front == -1){
front = 0;
}
rear++;
queue[rear] = data;
}
void dequeue () {
if (front == -1 || front > rear) {
printf("Queue underflow\n");
return ++;
}

void display() {
if (front == -1 || front > rear) {
printf("Queue is empty\n");
return;
}
printf("Queue elements are: \n");
for (int i = front; i <= rear; i++) {
printf("%d\n", queue[i];
}
}

int main () {
int choice, data;
while (1) {
```

```
printf("1. Enqueue\n 2. Dequeue\n 3. Display\n 4. Exit\n");
 printf("Enter your choice: ");
 scan("%d", & choice);
 switch (choice) {

 case 1:
 printf ("Enter data to be inserted:");
 scanf("%d, &data);
 enqueue (data);
 break;

 case 2;
 dequeue();
 break;
 case 3;
 display();
 break;
 case 4;
 exit(0);
 default;
 printf("invalid choice\n");
 }
 }
 return 0;
 }
```

Output:

1. Enqueue
2. Dequeue
3. Display
4. Exit

```
Enter your choice : 1
Enter data to be inserted : 54
1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice : 3
Queue elements are;
54
```

Result:

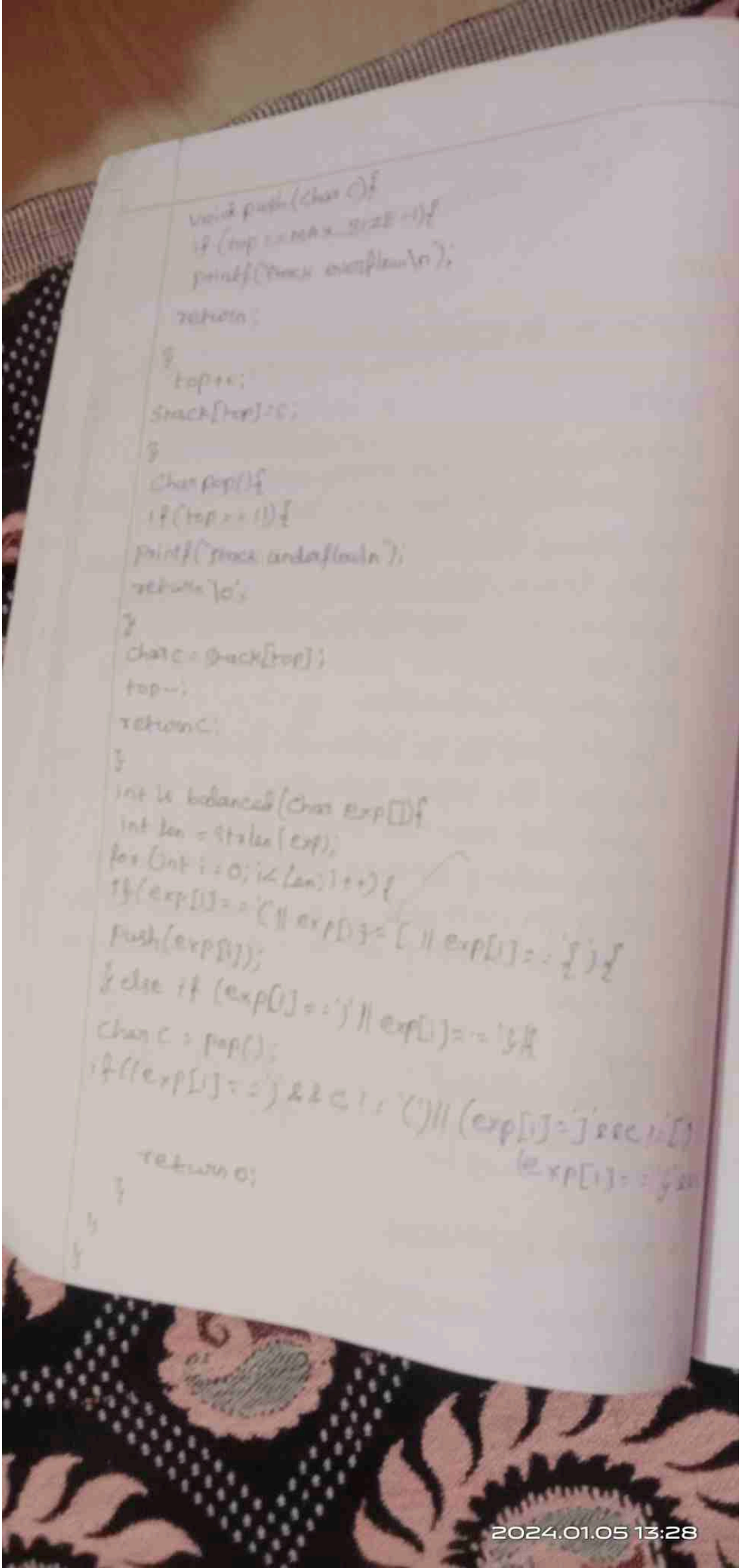
Thus the program is successfully implemented for array implemention of queue.

8. write a C program for balancing paranthEses using stack.

AIM:
C program for balancing paranthEses using Stack.

Algorithm:
1. Define a charater array to represent the stack and initialize the top of the stack to -1
2. Implement the push function to add charcters to the Stack, Checking for Stack overflow.
3. Implement the pop function to remove and return the top character from the stack, Checking for stack underflow.
4. Implement the is balanced function to iterate through the input expression and push opening parentheses onto the stack and pop the corresponding opening parenthesis when a closing parentheses is encountered. If the stack is empty at the end. the expression is balanced.

Coding:
```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define MAX-SIZE 100

char stack[MAX-SIZE];
int top=-1;
```



```
void push(char c){
if(top == MAX_SIZE-1){
printf("Stack overflow\n");
return;
}
top++;
stack[top]=c;
}
char pop(){
if(top == -1){
printf("Stack underflow\n");
return '\0';
}
char c = stack[top];
top--;
return c;
}
int is_balanced(char exp[]){
int len = strlen(exp);
for(int i=0; i<len; i++){
if(exp[i]=='(' || exp[i]=='[' || exp[i]=='{'){
push(exp[i]);
} else if (exp[i]==')' || exp[i]==']' || exp[i]=='}'){
char c = pop();
if((exp[i]==')' && c!='(') || (exp[i]==']' && c!='[')
                          || (exp[i]=='}' && c!='{'))
return 0;
}
}
}
```

```
    if(top == -1){
    return 1;
    }else{
     return 0;
    }
}

int main(){
 char exp[MAX_SIZE];
printf("Enter an expression:");
scanf("%s",exp);
if(is balanced(exp)){
 printf("The expression is balanced.\n");
}else{
 printf("The expression is not balanced.\n");
}
 return 0;
}
```

Output:

```
Enter an expression:((
The expression is not balanced.
```

Result:

Thus the program is sucessfully implemented for balancing the paranthesis in stack.

9. write a C program for Linked List implementation of Queue.

AIM:
C program for linked list implementation of Queue.

Algorithm:
1. Define a structure to represent a node in the linked list. Containing an integer value and a pointer to the next node.
2. Define pointers to the front and rear of the queue and initialize them to NULL.
3. Implement the Create function to initialize the queue.
4. Implement the enq function to add a new node to the rear of the queue.
5. Implement the deq function to remove the node from the front of the queue.
6. Implement the frontelement function to return the value of the node at the front of the queue.
7. Implement the display function to print the values of all nodes in the queue.
8. Implement a menu-driven main function to allow the user to interact with the queue.

Coding:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
  int info;
  struct node *ptr;
```

```
};
*front, *rear, *temp, *front1;
int frontelement();
void enq(int data);
void deq();
void display();
void create();
int count = 0;
void main()
{
 int no, ch, e;
printf("\n 1. enque\n 2. deque\n 3. front element\n 4. display");
create();
 while (1)
{
printf("\n Enter choice:");
scanf("%d", &ch);
switch (ch)
{
case 1:
printf("\n Enter data:");
scanf("%d", &no);
enq(no);
break;
case 2:
deq();
break;
```

```
case 3:
e = Frontelement();
if(e != 0)
printf("In Front element : %d", e);
else
printf("In No front element in Queue as Queue is
                                        empty");
break;
case 4:
display();
 break;
 }
}
}
void create()
{
front = rear = NULL;
}
void enq(int data)
{
if(rear == NULL)
{
rear = (struct node*)malloc(1*sizeof(struct node
rear->ptr = NULL;
rear->info = data;
front = rear;
}
else
{
temp = (struct node *)malloc(1*sizeof(struc
rear->ptr = temp;
```

```
temp->info = data;
temp->ptr = NULL;
rear = temp;
}
count++;
}
void display()
{
front1 = front;
if ((front1 == NULL) && (rear == NULL))
{ printf("\n Queue is empty");
return;
}
while (front1 != rear)
{
printf("%d", front1->info);
front1 = front1->ptr;
}
if (front1 == rear)
printf("%d", front1->info);
}
void deq()
{
front1 = front;
if(front1 == NULL)
{
printf("\n its an empty queue");
return;
}
else if front1->ptr1 = NULL)
```

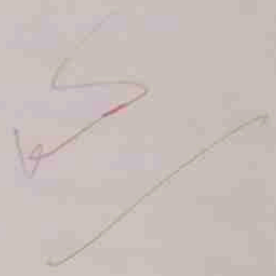
```
{ front1 = front1->ptr;
printf("\n Dequed value : %d", front->info);
free(front);
front = front1;
}
else
{ printf("\n Dequed value : %d", frond->info);
free(front);
front = NULL;
rear = NULL;
}
count--;
}
int frontelement()
{
if((front != NULL) && (rear != NULL))
return(front->info);
else
return 0;
}
```

Output:

1. enque
2. deque
3. front element
4. display

Enter Choice : 1
Enter data : 34
Enter Choice : 1
Enter data : 54

Enter choice : 2
Dequed value : 34



Result :
Thus the program is successfully implemented for linked list implementation of Queue.

10. write a C program foo polynomial addition using linked list.

AIM:
C program for polynomial addition using linked list.

Algorithm:
The program defines a struct node that contains an integer coefficient and an integer power, and a pointer to the next node. The program also defines functions to add two polynomials, display a polynomial, and create a node. The main function creates two polynomials, adds them, and displays the result.

Coding:

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

struct node{
  int coeff;
  int pow;
  struct node* next;
};

struct node* add poly(struct node *poly1, struct
struct node *result = NULL
struct node* tail = NULL
while (poly1 != NULL && poly2 != NULL){
if (poly1->pow > poly2->pow)
```

```
if (result == NULL){
 result = tail = poly1;
}else {
 tail -> next = poly1;
 tail = poly1;
}
 poly = poly1 ->next;
}else if(poly1 -> pow < poly2 -> pow){
 if (result == NULL){
   result = tail = poly2;
}else{
   tail -> next = poly2;
   tail = poly2;
}
 poly 2 = poly2 -> next;
}else{
 struct node *temp = (struct node *)malloc(sizeof(struct
                                                  node));
  temp -> coeff = poly1 -> Coeff + poly2 -> coeff;
   temp -> pow = poly1 -> pow;
    temp -> next = NULL;
    if (result == NULL){
      result = tail = temp;
}else{
 tail -> next = temp;
 tail = temp;
}
 poly1 = poly1 -> next;
 poly2 = poly2 -> next;
```

```
    }
    while (poly1 != NULL){
      if (result == NULL){
      result = tail = poly1;
    }else{
      tail -> next = poly1;
      tail = poly1;
    }poly1 = poly1 -> next;
    }
    while (poly2 != NULL){
    if (result == NULL){
     result = tail = poly2;
    }else{
     tail -> next = poly2;
     tail = poly2;
    }
    poly2 = poly2 -> next;
    }
    return result;
    }
    void display(struct node *poly){
    while (poly != NULL){
    printf("%dx^%d", poly -> coeff, poly -> pow
     poly = poly -> next;
    if (poly != NULL){
    printf("+");
      }
    }
```

```
printf("\n");
}
int main(){
struct node* poly1 =(struct node*)malloc(sizeof
                                (struct node));

struct node*poly2 =(struct node*)malloc(sizeof(struct node));
struct node *poly3 =(struct node*)malloc (sizeof(struct
                                node));
poly1-> Coeff =5;
poly1 -> pow = 2;
poly1->next =(struct node*) malloc (sizeof(struct node));
poly1 -> Coeff =5;
poly1-> pow = 2;
poly1 -> next=(struct node*) malloc (sizeof (struct node));
poly1-> next -> coeff =4;
poly1 -> next -> pow = 1;
poly1->next->next =(struct node*)malloc (sizeof (struct node));
poly->next->next->Coeff =2;
poly1->next->next -> pow =0;
poly1->next->next->next =NULL;
poly1->next->next->next->next=
Poly 2-> Coeff =-5;
poly2 ->Dow = 1;
poly2->next=(struct node*) malloc(sizeof(struct node));
poly.2->next-> Coeff = -5;
Poly 2->next-> pow = 0;
```

```
poly2->next->next=NULL;
printf("first polynomial:");
display(poly1);
printf("second pollynomial:");
display(poly2);
poly3=addpoly(poly1,poly2);
printf("Resultant Polynomial:");
display(poly3);

return 0;
}
```

Output:

First polynomial: $5x^2+4x^1+2x^0$

Second polynomial: $-5x^1+-5x^0$

Resultant polynomial: $5x^2+-1x^1+-3x^0$

Result:

Thus the program is successfully implemented for polynomial addition using linked list.