## Laboratory 6

1. Questions

    1. Implement the INSERT, DELETE and PRINT operations on queue.
    2. Implement a priority queue using suitable application.

2. Algorithm:-

**Q1:-**

**Insert operation:-**

- **Step 1** − Check if the queue is full.
- **Step 2** − If the queue is full, prints” queue is full”
- **Step 3** − If the queue is not full, increment **rear** pointer to point the next empty space.
- **Step 4** − Add data element to the queue location, where the rear is pointing.

**Delete operation:-**

- **Step 1** − Check if the queue is empty.
- **Step 2** − If the queue is empty, prints” queue is empty”
- **Step 3** − If the queue is not empty, access the data where **front** is pointing.
- **Step 4** − Increment **front** pointer to point to the next available data element.

**Print operation:-**

- **Step 1** − Check if the queue is empty.
- **Step 2** − f the queue is empty, prints” queue is empty”
- **Step 3** − If the queue is not empty, print the elements of stack by running ‘for loop’ from front to rear.

**Main function :-**

- **Step 1** – run do while loop if choice not eual to 4.
- **Step 2** – print "1: insert 2:delete 3:print 4:exit " and input int variable choice

**Step 3** – make use of switch(choice) statement to perform step 2 cases and hence call the respective functions.

**Q2:-**

**Algorithm :**

```
PUSH(HEAD, DATA, PRIORITY)
Step 1: Create new node with DATA and PRIORITY
Step 2: Check if HEAD has lower priority. If true follow Steps 3-4 and end. Else goto
        Step 5.

Step 3: NEW -> NEXT = HEAD
Step 4: HEAD = NEW
Step 5: Set TEMP to head of the list
Step 6: While TEMP -> NEXT != NULL and TEMP -> NEXT -> PRIORITY > PRIORITY
Step 7: TEMP = TEMP -> NEXT

[END OF LOOP]

Step 8: NEW -> NEXT = TEMP -> NEXT
Step 9: TEMP -> NEXT = NEW
Step 10: End
```

<u>POP(HEAD)</u>

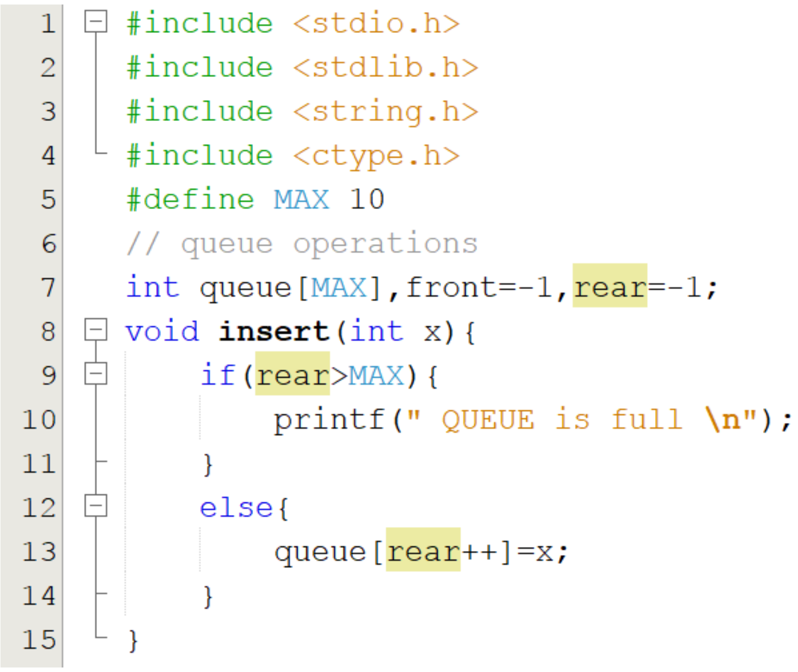
```
Step 2: Set the head of the list to the next node in the list. HEAD = HEAD -> NEXT.
Step 3: Free the node at the head of the list
Step 4: End
```

<u>PEEK(HEAD):</u>

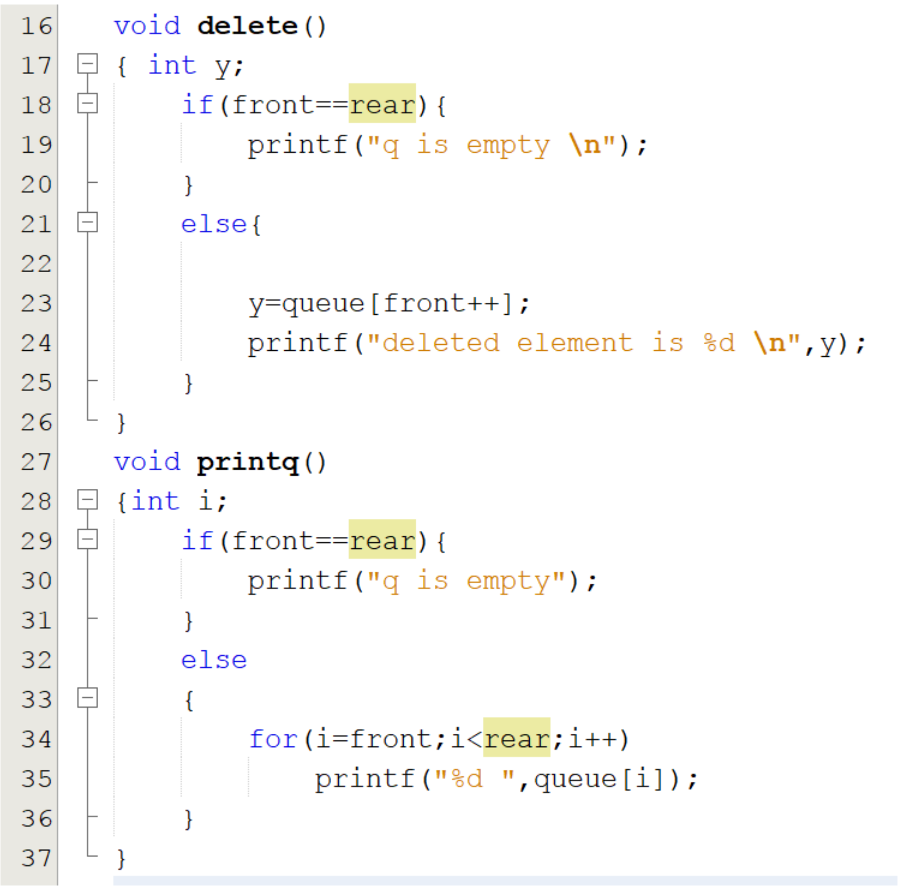
```
Step 1: Return HEAD -> DATA
Step 2: End
```

3. Program

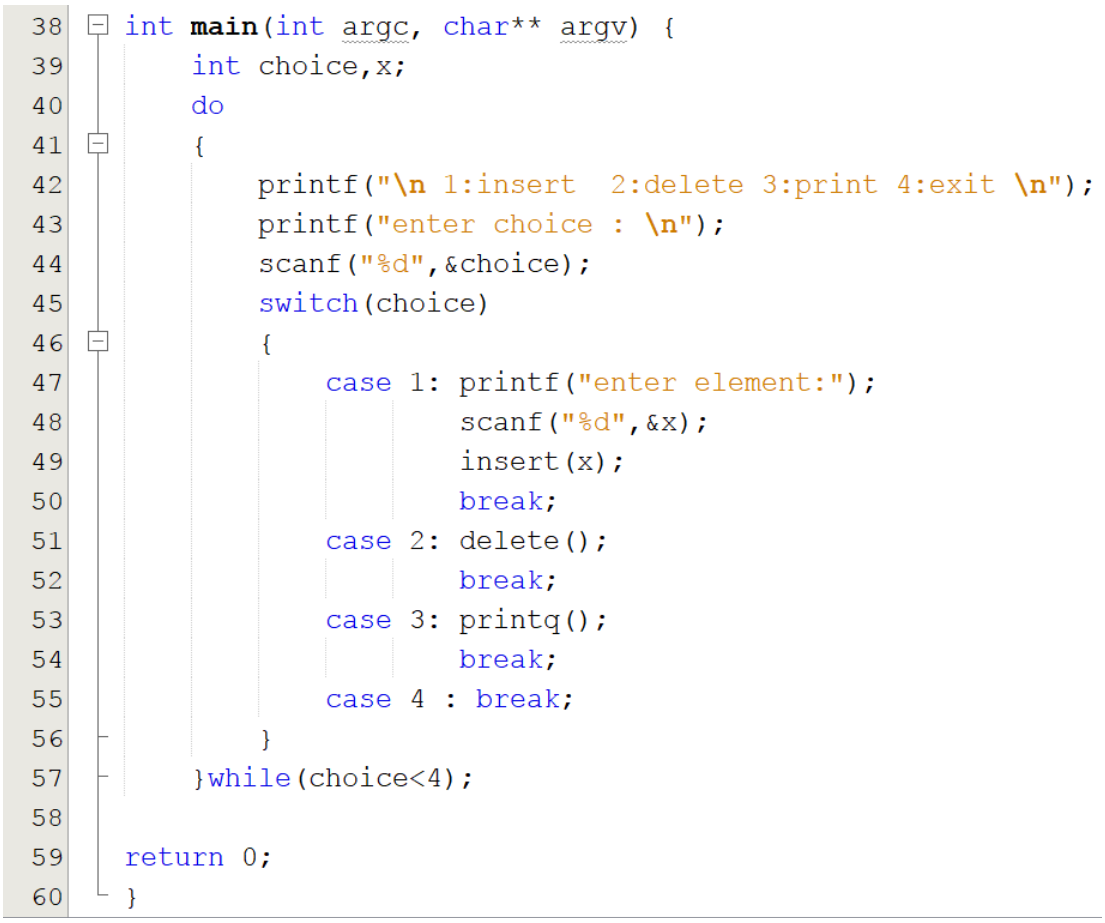
**Q1:** Implement the INSERT, DELETE and PRINT operations on queue.



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX 10
// queue operations
int queue[MAX],front=-1,rear=-1;
void insert(int x){
    if(rear>MAX){
        printf(" QUEUE is full \n");
    }
    else{
        queue[rear++]=x;
    }
}
```

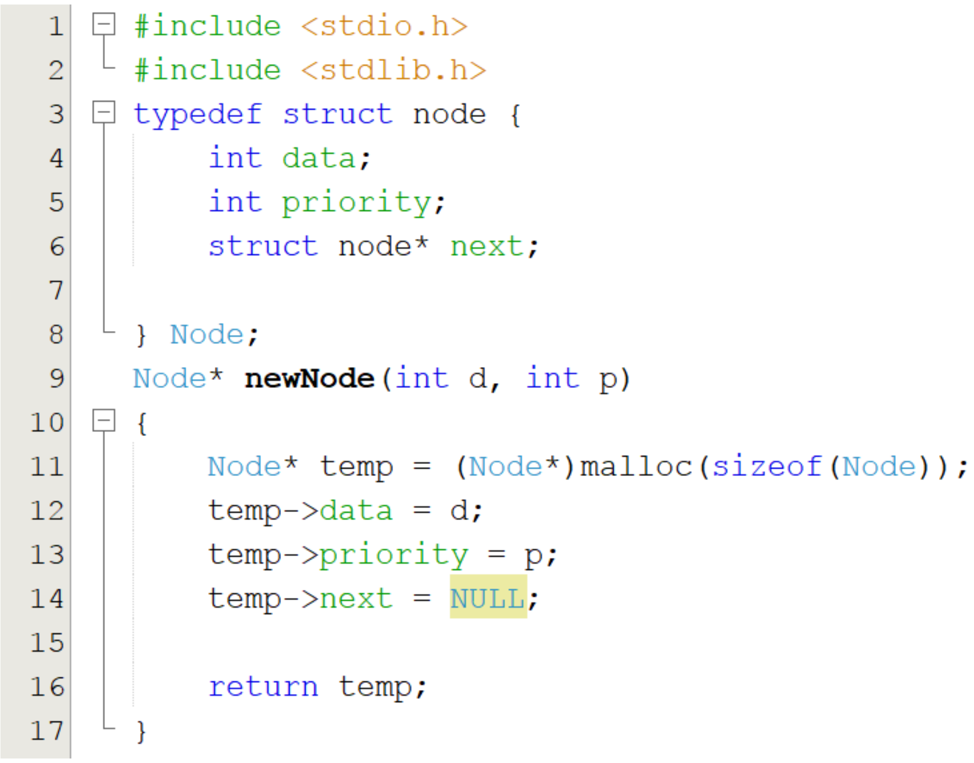


```c
void delete()
{ int y;
    if(front==rear){
        printf("q is empty \n");
    }
    else{

        y=queue[front++];
        printf("deleted element is %d \n",y);
    }
}
void printq()
{int i;
    if(front==rear){
        printf("q is empty");
    }
    else
    {
        for(i=front;i<rear;i++)
            printf("%d ",queue[i]);
    }
}
```

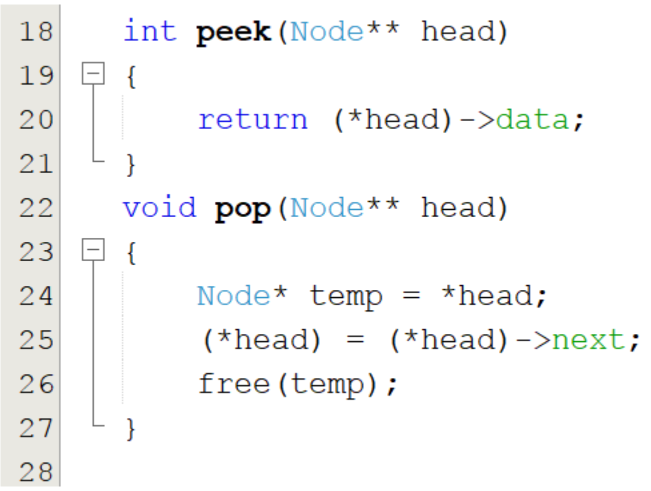


```
int main(int argc, char** argv) {
    int choice,x;
    do
    {
        printf("\n 1:insert  2:delete 3:print 4:exit \n");
        printf("enter choice : \n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: printf("enter element:");
                    scanf("%d",&x);
                    insert(x);
                    break;
            case 2: delete();
                    break;
            case 3: printq();
                    break;
            case 4 : break;
        }
    }while(choice<4);

return 0;
}
```

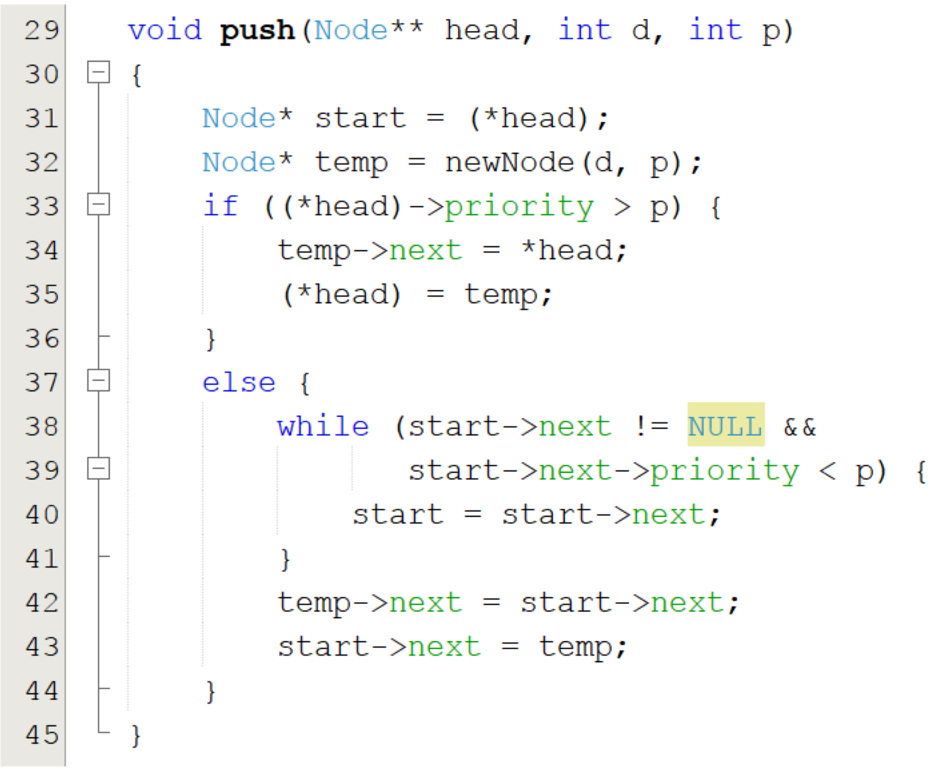
Q2:-



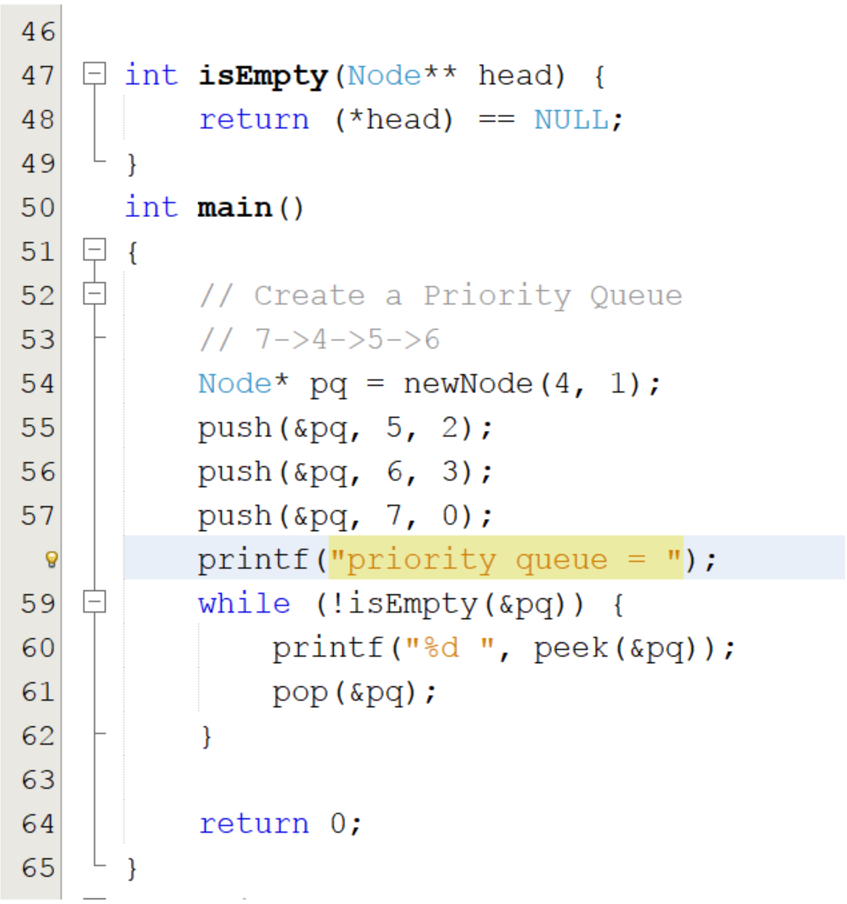
```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    int data;
    int priority;
    struct node* next;

} Node;
Node* newNode(int d, int p)
{
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = d;
    temp->priority = p;
    temp->next = NULL;

    return temp;
}
```



```c
int peek(Node** head)
{
    return (*head)->data;
}
void pop(Node** head)
{
    Node* temp = *head;
    (*head) = (*head)->next;
    free(temp);
}

```



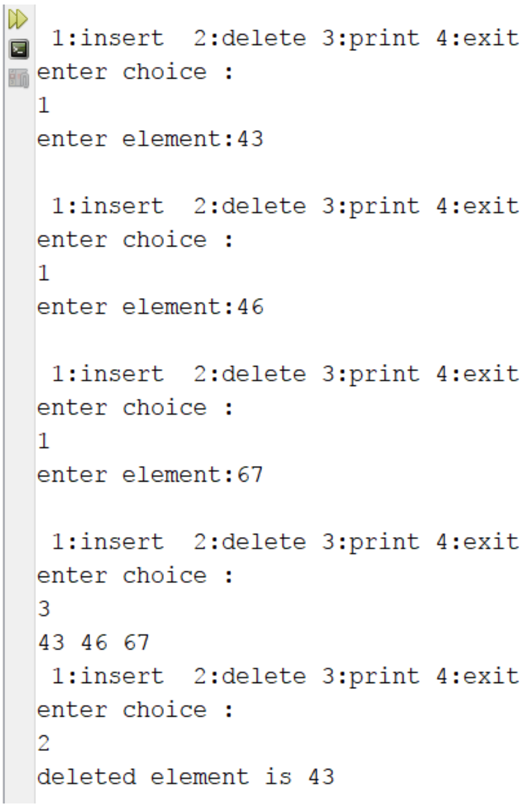
```c
void push(Node** head, int d, int p)
{
    Node* start = (*head);
    Node* temp = newNode(d, p);
    if ((*head)->priority > p) {
        temp->next = *head;
        (*head) = temp;
    }
    else {
        while (start->next != NULL &&
               start->next->priority < p) {
            start = start->next;
        }
        temp->next = start->next;
        start->next = temp;
    }
}
```



```

int isEmpty(Node** head) {
    return (*head) == NULL;
}
int main()
{
    // Create a Priority Queue
    // 7->4->5->6
    Node* pq = newNode(4, 1);
    push(&pq, 5, 2);
    push(&pq, 6, 3);
    push(&pq, 7, 0);
    printf("priority queue = ");
    while (!isEmpty(&pq)) {
        printf("%d ", peek(&pq));
        pop(&pq);
    }

    return 0;
}
```

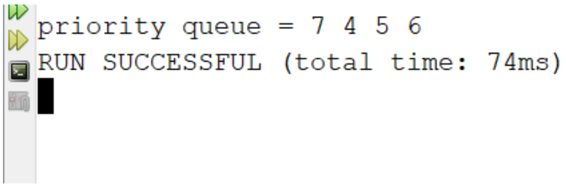
4. Presentation of Results

**Ans 1:** Implement the INSERT, DELETE and PRINT operations on queue:-

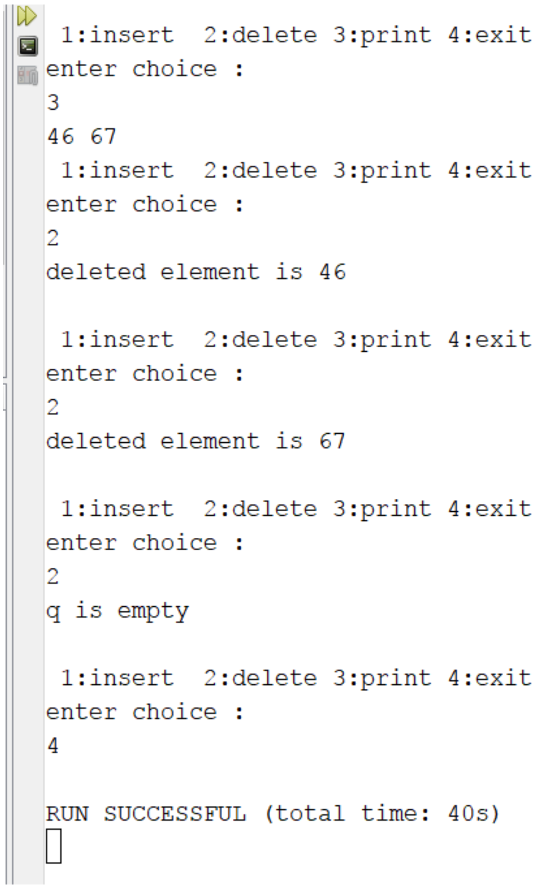


```
 1:insert  2:delete 3:print 4:exit
enter choice :
1
enter element:43

 1:insert  2:delete 3:print 4:exit
enter choice :
1
enter element:46

 1:insert  2:delete 3:print 4:exit
enter choice :
1
enter element:67

 1:insert  2:delete 3:print 4:exit
enter choice :
3
43 46 67
 1:insert  2:delete 3:print 4:exit
enter choice :
2
deleted element is 43
```

**Ans 2:-**



```
priority queue = 7 4 5 6
RUN SUCCESSFUL (total time: 74ms)
```



```
 1:insert  2:delete 3:print 4:exit
enter choice :
3
46 67
 1:insert  2:delete 3:print 4:exit
enter choice :
2
deleted element is 46

 1:insert  2:delete 3:print 4:exit
enter choice :
2
deleted element is 67

 1:insert  2:delete 3:print 4:exit
enter choice :
2
q is empty

 1:insert  2:delete 3:print 4:exit
enter choice :
4

RUN SUCCESSFUL (total time: 40s)
```

5. Conclusions :-

   All the programs have been executed successfully.