



Parul[®]
University

NAAC A++

**FACULTY OF ENGINEERING AND
TECHNOLOGY
BACHELOR OF TECHNOLOGY**

योगः कर्मसु कौशलम्

**Computational Thinking for Structured Design – 2
(303105151)**

**II SEMESTER
Computer Science & Engineering
Department**

PARUL UNIVERSITY

Laboratory Manual



FACULTY OF ENGINEERING & TECHNOLOGY
Computational Thinking for
Structured Design – 2
(303105151)
B.Tech - 1st Year 2nd Semester

CERTIFICATE

This is to certify that Mr./Ms. HARSHAL SHIRSATH has completed all the practical of the subject on his/her own and is submitting the lab manual with proper formatting as per the given instructions. His/Her work is found satisfactory and hence, he/she has successfully completed the term-work submission of **Computational Thinking for Structured Design-2 (303105151)**.



Date of Submission:

Staff In charge:

Head of Department:



Faculty of Engineering & Technology (FET)

Parul Institute of Engineering & Technology (PIET)
Department of Computer Science & Engineering

Practical Assessment Table

Sr. No.	Practical Title	Page No.		Mark (10)	Sign
		From	To		
1	1.1 Write a c program to increase or decrease the existing size of an 1D array. 1.2 Write a c program on 2D array to Increase & Decrease i) No of subarrays ii) elements in the subarrays				
2	2.1 Write a to display present date and time using c language. 2.2 Write a c program to demonstrate pre-processor directives i) Macros ii) Conditional Compilation				
3	1. Write a C program that uses functions to perform the following Operations. i) Reading a complex number ii) Writing a complex number iii) Addition of two complex numbers iv) Multiplication of two complex numbers 2. Write a c program to store records of n students based on roll_no, name, gender and 5 subject marks i) Calculate the percentage of each student using 5 subjects. ii) Display the student list according to their percentages.				
4	Write a C program to store n employee records based on EMP_ID,EMP_NAME,EMP_DEPTID,EMP_PHNO,E				

	MP_SALARY and display all the details of employees using EMP_NAME in sorted order.				
5	<p>1. Write a c program to implement selection Sort & Bubble sort</p> <p>2. Write a C program to reverse the elements within a given range in a sorted list.</p> <p>Example :</p> <p>input : 10 9 1 2 4 3 4 6 7 8 10 3 8</p> <p>output: 1 2 8 7 6 4 4 3 9 10</p> <p>the sorted list of given array elements is 1 2 3 4 4 6 7 8 9 10 ,</p> <p>after reversing the elements within the range 3 and 8 is 1 2 8 7 6 4 4 3 9 10</p>				
6	<p>1. Write a c program to implement Insertion sort & Quick sort</p> <p>2. Write a c program to sort the given n integers and perform following operations</p> <p>i) Find the products of every two odd position elements</p> <p>ii) Find the sum of every two even position elements</p> <p>Explanation:</p> <p>Input : 9 1 9 8 3 5 4 7 2 6</p> <p>Output: 3 15 35 63 6 10 14</p> <p>The sorted list of given input is 1 2 3 4 5 6 7 8 9,</p> <p>the product of alternative odd position elements is $1*3 = 3, 3*5=15, 5*7=35...$</p> <p>and the sum of two even position elements $2+4 = 6, 4+6=10$.</p>				
7	Write a C Program to implement Merge Sort.				
8	<p>1. Write a c program to sort in ascending order and reverse the individual row elements of an mxn matrix</p> <p>input :</p> <p>3 4</p> <p>1 4 2 3</p> <p>7 8 10 9</p> <p>6 3 5 2</p> <p>output:</p> <p>4 3 2 1</p> <p>10 9 8 7</p> <p>6 5 3 2</p> <p>2. Write a c program to sort elements in row wise and print the elements of matrix in Column major order</p> <p>Input:</p> <p>3 4</p>				

	1 4 2 3 7 8 10 9 6 3 5 2 Output: 1 7 2 2 8 3 3 9 5 4 10 6 Explanation: The sorted matrix according to the conditions is 1 2 3 4 7 8 9 10 2 3 5 6 after sorting matrix the elements as to be printed in column major order 1 7 2 2 8 3 3 9 5 4 10 6				
9	1. Write a c program to perform linear Search. 2. Write a c program to perform binary search.				
10	Write a c program to Create a single Linked list and perform Following Operations A.Insertion At Beginning B. Insertion At End C.Insertion After a particular node D.Insertion Before a particular node E. Insertion at specific position F.Search a particular node G.Return a particular node H. Deletion at the beginning I. Deletion at the end J.Deletion after a particular node K.Deletion before a particular node L.Delete a particular node M.Deletion at a specific position				
11	1. Write a program to Reverse a singly Linked list. 2. Write a c program to check whether the created linked list is palindrome or not				

12	Write a c program to Create a Circular Linked list and perform Following Operations A.Insertion At Beginning B.Insertion At End C.Insertion After a particular node D.Insertion Before a particular node E.Insertion at specific position F.Search a particular node G.Return a particular node H.Deletion at the beginning I. Deletion at the end J.Deletion after a particular node K.Deletion before a particular node L.Delete a particular node M.Deletion at a specific position				
13	Write a c program to Create a Circular single Linked list and perform Following Operations A.Insertion After a particular node B.Insertion Before a particular node C.Search a particular node D.Return a particular node E.Deletion before a particular node F.Delete a particular node				
14	Write a c program to Create a Circular Doubly Linked list and perform Following Operations A.Insertion After a particular node B.Insertion Before a particular node C.Search a particular node D.Return a particular node E.Deletion before a particular node F.Delete a particular node				

Practical-1

Aim-1.1:- Write a c program to increase or decrease the existing size of a 1D array.

Theory:- This program asks the user for the initial size of the array, then allows them to input elements. After that, it asks for the new size of the array and resizes it accordingly using `realloc()`. Finally, it prints out the elements of the resized array.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
    int *p,n,i,newlength;
    printf("enter the size of array\n");
    scanf("%d",&n);
    p=(int*)malloc(n*sizeof(int));
    if(p==NULL)
    {
        printf("memory allocate fail");
        exit(1);
    }
    printf("enter the values\n");
    for(i=0;i<n;i++)
        scanf("%d",&p[i]);
    printf("values are: \n", p[i]);
    for(i=0;i<n;i++){
        printf("%d ",p[i]);
        printf("\nenter new size\n");
        scanf("%d",&newlength);
        for(i=0;i<newlength;i++){
            p=realloc(p,n*sizeof(int));
            printf("enter values\n");
            for(i=0;i<newlength;i++)
                scanf("%d",&p[i]);
            printf("values are: ", p[i]);
            for(i=0;i<newlength;i++)
                printf("%d ",p[i]); free(p);
        }
    }
```

Output:-

```
enter the size of array
3
enter the values
2
2
2
values are:
2 2 2
enter new size
3
enter values
1
1
1
values are: 1 1 1
-----
Process exited after 33.28 seconds with return value 1
Press any key to continue . . .
```


Aim1.2:- Write a c program on 2D array to Increase & Decrease**i) No of subarrays ii) elements in the subarrays**

Theory:- This program prompts the user for the initial number of subarrays and elements in each subarray, input the elements, then asks for the new number of subarrays and elements in each subarray. It reallocates memory for the 2D array accordingly and prints the updated array.

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

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

```
int main(int argc, char const *argv[]) { int
```

```
    N,M;
```

```
    int count = 0;
```

```
    int **A;
```

```
    printf ("Enter the size for first array: ");
```

```
    scanf("%d",&N);
```

```
    A = (int **)malloc(sizeof(int *)*N); for(int
```

```
    i=0; i<N; ++i){
```

```
        *(A+i) = (int *)malloc(sizeof(int)*N);
```

```
    }
```

```
    for(int i=0; i<N; ++i){
```

```
        for(int j=0; j<N; ++j){
```

```
            A[i][j] = count++;
```

```
        }
```

```
    }
```

```
    for(int i=0; i<N; ++i){
```

```
        for(int j=0; j<N; ++j){
```

```
            printf("%2d ", A[i][j]);
```

```
        }
```

```
        printf("\n");
```

```
    }
```

```
    printf ("Enter the size for new array: ");
```

```
    scanf("%d",&M);
```

```
    A = (int **)realloc(A, sizeof(int **)*M);
```

```
    for(int i=0; i<N; ++i){
```

```
        *(A+i) = (int *)realloc(*(A+i), sizeof(int)*M);
```

```
    }
```

```
    for(int i=N; i<M; ++i){
```

```
        *(A+i) = (int *)malloc(sizeof(int)*M);
```

```
    }
```

```
    for(int i=0; i<N; ++i){
```

```
        for(int j=N; j<M; ++j){
```

```
            *(*A+i)+j = 0;
```

```
    }  
}  
  
for(int i=N; i<M; ++i){  
    for(int j=0; j<M; ++j){  
        *(*(A+i)+j) = 9;  
    }  
}  
  
for(int i=0; i<M; ++i){  
    for(int j=0; j<M; ++j){  
        printf("%2d ", A[i][j]);  
    }  
    printf("\n");  
}  
  
return 0;  
}
```

Output:

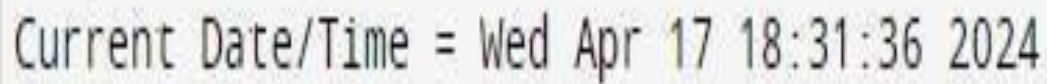
```
Enter the size for first array: 3  
0  1  2  
3  4  5  
6  7  8  
Enter the size for new array: 3  
0  1  2  
3  4  5  
6  7  8  
  
=== Code Execution Successful ===
```

Practical:-2**Aim-2.1:- Write a to display present date and time using c language**

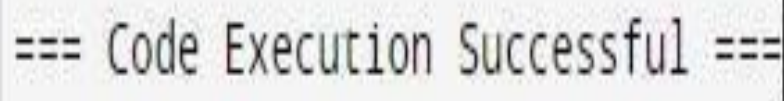
Theory:- This program will display the current date and time in the format **YYYY-MM-DD HH:MM:SS**.

```
#include<stdio.h>
#include<time.h> int
main()
{
    time_t tm;
    time(&tm);
    printf("Current Date/Time = %s", ctime(&tm));
    return 0;
}
```

Output:-



Current Date/Time = Wed Apr 17 18:31:36 2024



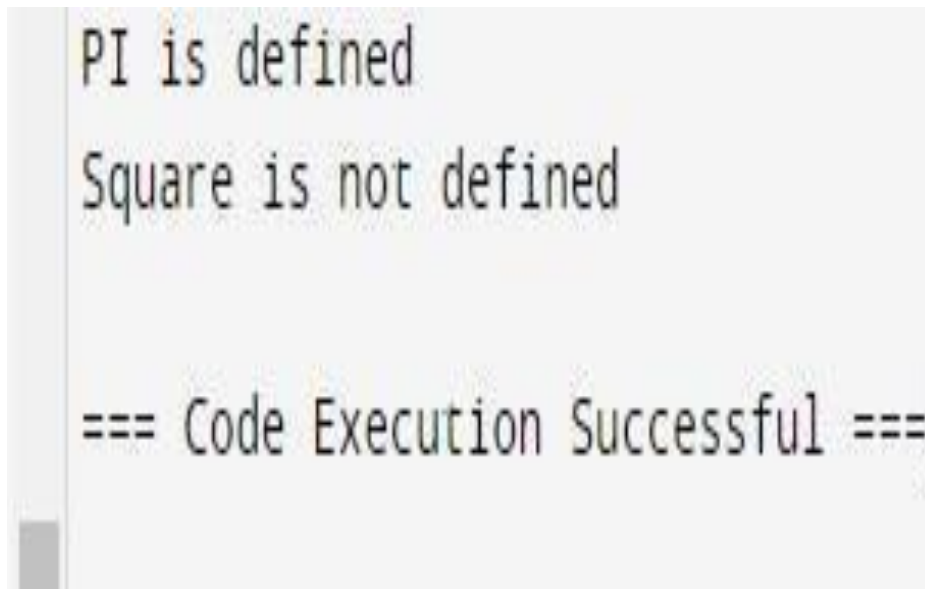
=== Code Execution Successful ===

Aim:2.2:- Write a c program to demonstrate pre-processor directives**i) Macros****ii) Conditional Compilation**

```
#include<stdio.h>
#define PI 3.14159
int main(){
#ifdef PI
    printf("PI is defined\n");
#elif defined(SQUARE)
    printf("Square is defined\n");

#else
#error "Neither PI nor SQUARE is defined" #endif
#endif SQUARE
    printf("Square is not defined");

#else
printf( "Square is defined" );
#endif
    return 0;
}
```

Output:-

```
PI is defined
Square is not defined

=== Code Execution Successful ===
```

Practical :-3

Aim:3.1:- Write a C program that uses functions to perform the following Operations.

- 1. Reading a complex number**
- 2. Writing a complex number**
- 3. Addition of two complex numbers**
- 4. Multiplication of two complex numbers**

```
#include <stdio.h>
#include <conio.h>
struct complex {
    float real, imag;
} a, b, c;
struct complex read(void);
void write(struct complex);
struct complex add(struct complex, struct complex);
struct complex sub(struct complex, struct complex);
struct complex mul(struct complex, struct complex);
struct complex div(struct complex, struct complex);
void main () {
    clrscr();
    printf("Enter the 1st complex number\n "); a =
    read();
    write(a);
    printf("Enter the 2nd complex number\n"); b =
    read();
    write(b);
    printf("Addition\n "); c
    = add(a, b);
    write(c);
    printf("Substraction\n "); c
    = sub(a, b);
    write(c);
    printf("Multiplication\n"); c
    = mul(a, b);
    write(c);
    printf("Division\n"); c
    = div(a, b);
    write(c);
    getch();
}
struct complex read(void) { struct
complex t;
printf("Enter the real part\n");
scanf("%f", &t.real);
printf("Enter the imaginary part\n");
scanf("%f", &t.imag);
return t;
}
void write(struct complex a)
{
    printf("Complex number is\n"); printf("
%.1f + i %.1f", a.real, a.imag);
    printf("\n");
}
```

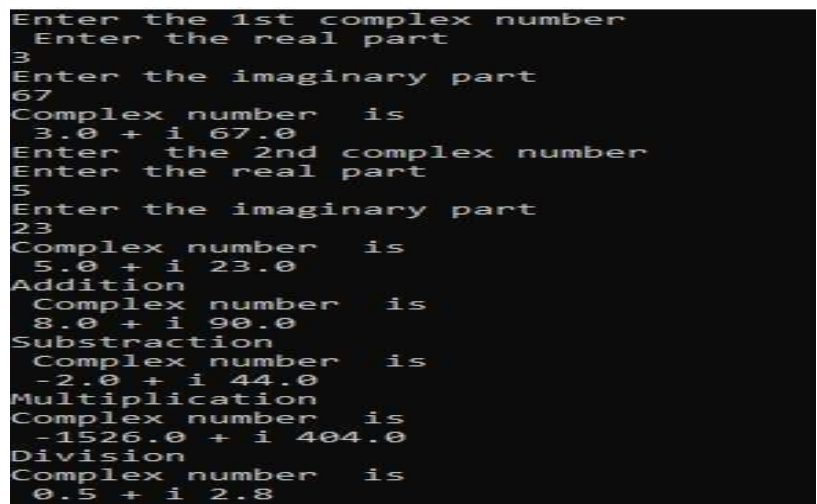
```
struct complex add(struct complex p, struct complex q)
{
    struct complex t;
    t.real = (p.real + q.real);
    t.imag = (p.imag + q.imag);
    return t;
}

struct complex sub(struct complex p, struct complex q)
{
    struct complex t;
    t.real = (p.real - q.real);
    t.imag = (p.imag - q.imag);
    return t;
}

struct complex mul(struct complex p, struct complex q)
{
    struct complex t;
    t.real=(p.real * q.real) - (p.imag * q.imag);
    t.imag=(p.real * q.imag) + (p.imag * q.real);
    return t;
}

struct complex div(struct complex p, struct complex q)
{
    struct complex t;
    t.real = ((p.imag * q.real) - (p.real * q.imag)) / ((q.real * q.real) + (q.imag * q.imag)); t.imag =
    ((p.real * q.real) + (p.imag * q.imag)) / ((q.real * q.real) + (q.imag * q.imag)); return(t);
}
```

Output:-



```
Enter the 1st complex number
Enter the real part
3
Enter the imaginary part
67
Complex number is
3.0 + i 67.0
Enter the 2nd complex number
Enter the real part
5
Enter the imaginary part
23
Complex number is
5.0 + i 23.0
Addition
Complex number is
8.0 + i 90.0
Substraction
Complex number is
-2.0 + i 44.0
Multiplication
Complex number is
-1526.0 + i 404.0
Division
Complex number is
0.5 + i 2.8
```

Aim:3.2:- Write a c program to store records of n students based on roll_no, name, gender and 5 subject marks i) Calculate the percentage of each student using 5 subjects. ii) Display the student list according to their percentages.

Theory:-

- The program uses a structure `student` to represent each student, including their roll number, name, gender, marks in five subjects, and percentage.
- It calculates the percentage for each student based on their marks in five subjects.
- Then, it sorts the students based on their percentages using `qsort`.
- Finally, it displays the student list sorted by percentage.

```
#include<stdio.h> int
main(){
struct student    {

        char firstname[50];
        int roll;
        char gender[6];

        float total,percentage;

}s[5];

        int i;
float  math; float
physics; float
chemistry; float
english; float
hindi;
float percentage,total;

printf("enter information of students:\n");
for(i=0;i<=1;i++)
{

        s[i].roll=i+1;

        printf("for roll number %d,\n",s[i].roll);
        printf("enter first name:");
        scanf("%s",s[i].firstname); printf("enter
gender:"); scanf("%s",s[i].gender);
        printf("enter marks for math:");
        scanf("%f",&math);
        printf("enter marks for physics:");
        scanf("%f",&physics);
        printf("enter marks for chemistry:");
        scanf("%f",&chemistry); printf("enter
marks for english:");
```

```
scanf("%f",&english);
printf("enter marks of hindi");
scanf("%f",&hindi);
s[i].total=math+physics+chemistry+english+hindi;
total=math+physics+chemistry+english+hindi;
s[i].percentage=(total/500)*100;
}

printf("\ndisplaying
informations:\n\n");for(i=0;i<=1;i++){
    printf("\n roll number is %d\n",i+1);
    printf("firstname:"); puts(s[i].firstname);
    printf("marks:%.1f",s[i].total);
    printf("\npercentage is %.2f\n",s[i].percentage);
    printf("\n");
}
return 0;
}
```

Output:-

```
enter information of students:
for roll number 1,
enter first name:Aman
enter gender:male
enter marks for math:99
enter marks for physics:98
enter marks for chemistry:98
enter marks for english:99
enter marks of hindi100
for roll number 2,
enter first name:Ankita
enter gender:female
enter marks for math:80
enter marks for physics:98
enter marks for chemistry:97
enter marks for english:100
enter marks of hindi80

displaying informations:

roll number is 1
firstname:Aman
marks:494.0
percentage is 98.80

roll number is 2
firstname:Ankita
marks:455.0
percentage is 91.00

-----
Process exited after 99.35 seconds with return value 0
Press any key to continue . . .
```


Practical:-4

Aim:4.1:- Write a C program to store n employee records based on EMP_ID,EMP_NAME,EMP_DEPTID,EMP_PHNO,E MP_SALARY and display all the details of employees using EMP_NAME in sorted order.

Theory:- This program uses an array of structures to store the employee records. It prompts the user to enter the details for each employee, including EMP_ID, EMP_NAME, EMP_DEPTID, EMP_PHNO, and EMP_SALARY. Then, it sorts the records based on EMP_NAME using the `qsort` function from the standard library, and finally, it displays all the employee details in sorted order.

```
#include<stdio.h>
int main()
{
    struct employee
    {
        int salary,id,p_no,dept_id;
        char name[100];
    }s[100],s1[100];
    int n,i,j,index=0;
    printf("Enter number total no. of Employee=\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Employee[%d]-id=\n",i);
        scanf("%d",&s[i].id);

        printf("Employee[%d]-name=\n",i);
        scanf("%s",s[i].name);

        printf("Employee[%d]- phone number=\n",i);
        scanf("%d",&s[i].p_no);

        printf("Employee[%d]-dept id=\n",i);
        scanf("%d",&s[i].dept_id);

        printf("Employee[%d]-salary=\n",i);
        scanf("%d",&s[i].salary);
    }

    for(i=0;i<=n;i++)
    {
        s1[i]=s[i];
    }

    printf("\nID\tNAME\tSALARY\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t%s\t%d\n",s[i].id,s[i].name,s[i].salary);
    }
}
```

```

}
for(i=0;i<n;i++) {
    for(j=0;j<n-1-i;j++) {
        if(s[j].salary<s[j+1].salary) { s[j].salary=s[j].salary
            ^ s[j+1].salary; s[j+1].salary=s[j].salary ^
            s[j+1].salary; s[j].salary=s[j].salary ^
            s[j+1].salary;
        }
    }
}
printf("Sorted By Salary is=\n\n");
printf("\nID\tNAME\tSALARY\n");
for(i=0;i<n;i++) {
    for(j=0;j<n;j++) {
        if(s[i].salary==s1[j].salary)
            index=j;
    }
    printf("%d\t%s\t%d\n",s1[index].id,s1[index].name,s1[index].salary);
}
return 0;
}

```

Output:-

```

Employee[0]- phone number=
9658741236
Employee[0]-dept id=
94263363
Employee[0]-salary=
1000000
Employee[1]-id=
654321
Employee[1]-name=
Abhinav
Employee[1]- phone number=
986321478
Employee[1]-dept id=
654238
Employee[1]-salary=
10000

ID      NAME      SALARY
123456  Aman      1000000
654321  Abhinav  10000
Sorted By Salary is=

ID      NAME      SALARY
123456  Aman      1000000
654321  Abhinav  10000

-----
Process exited after 108.7 seconds with return value 0
Press any key to continue . . .

```

Practical:-5

Aim:5.1:- Write a c program to implement selection Sort & Bubble sort 2. Write a C program to reverse the elements within a given range in a sorted list.

Example :

input : 10 9 1 2 4 3 4 6 7 8 10 3 8

output: 1 2 8 7 6 4 4 3 9 10

the sorted list of given array elements is 1 2 3 4 4 6 7 8 9 10 , after reversing the elements within the range 3 and 8 is 1 2 8 7 6 4 4 3 9 10

Theory :- This program first sorts the array using Selection Sort, then reverses the elements within the range [3, 8] using the **reverseRange** function, and finally prints the sorted and reversed array.

```
#include <stdio.h>
#include <stdbool.h>
void swap(int *xp, int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
void bubbleSort(int arr[], int n){
    int i, j;
    bool swapped;
    for (i = 0; i < n - 1; i++) {
        swapped = false;
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(&arr[j], &arr[j + 1]);
                swapped = true;
            }
        }
        if (swapped == false)
            break;
    }
}
void selectionSort(int arr[], int n){ int
    i, j, min_idx;
    for (i = 0; i < n-1; i++){
        min_idx = i;
        for (j = i+1; j < n; j++) if
            (arr[j] < arr[min_idx])
                min_idx = j;
        if(min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}
void printArray(int arr[], int size){
    int i;

    for (i=0; i < size; i++)
```

```
printf("%d ", arr[i]);
printf("\n");
}
int main(){
int arr[10],arr1[10],N,M;
printf("Enter the number for selection: "); scanf
("%d",&N);
for(int i = 0; i < N; ++i) {
scanf("%d", &arr[i]);
}
printf("Displaying integers: \n");
for(int i = 0; i < N; ++i) {
printf("%d\n", arr[i]);
}
printf("Enter the number for bubble: ");
scanf ("%d",&M);
for(int i = 0; i < M; ++i) {
scanf("%d", &arr1[i]);
}
printf("Displaying integers: \n");
for(int i = 0; i < N; ++i) {
printf("%d\n", arr1[i]);
} int n = sizeof(arr)/sizeof(arr[0]);
selectionSort(arr, n);
bubbleSort(arr1, n); printf("Sorted
array: \n"); printArray(arr, n);
printArray(arr1, n);
return 0;
}
```

Output:-

```
Enter the number for selection: 4
6
58
9
8
Displaying integers:
6
58
9
8
Enter the number for bubble: 4
6
58
9
8
Displaying integers:
6
58
9
8
Sorted array:
0 0 0 0 0 6 8 9 17 58
-1 -1 0 0 1 6 8 9 58 1708880
-----
Process exited after 40.76 seconds with return value 0
Press any key to continue . . .
```

Practical:-6**Aim:6.1:- 1. Write a c program to implement Insertion sort & Quick sort**

Theory:- Insertion Sort: Builds the sorted array gradually, inserting each element into its correct position. Quick Sort: Efficiently partitions the array based on a chosen pivot element, recursively sorting subarrays.

```
#include<stdio.h>
void swap(int* p1,int* p2){
    int temp;
    temp=*p1;
    *p1=*p2;
    *p2=temp;
}
void insertionSort(int arr[],int n){ int
    i, key,j;
    for(i=1;i<n;i++){
        key=arr[i];
        j=i-1;
        while (j>=0 && arr[j]>key){
            arr[j+1]=arr[j];
            j=j-1;
        }
        arr[j+1]=key;
    }
}
int partition(int arr[] , int low,int high){ int
    pivot = arr[high];
    int j,i =(low-1);
    for( j=low;j<=high;j++){
        if(arr[j]<pivot){
            i++;
            swap(&arr[i],&arr[j]);
        }
    }
    swap(&arr[i+1], &arr[high]);
    return (i+1);
}
void quickSort(int arr[],int low, int high){
    if(low<high){
        int pi= partition (arr,low,high);
        quickSort(arr,low,pi-1);
        quickSort(arr,pi+1,high);
    }
}
void printArray(int arr[],int n){ int
    i;
    for(i=0;i<n;i++)
        printf("%d ",arr[i]);
    printf("\n");
}
int main(){
    int arr[7],arr1[7],M,N,i;
    printf("Enter the size for quick sort:");
    scanf("%d",&N);
    for(i=0;i<N;i++){
```

```
        scanf("%d",&arr[i]);
    }
    printf("%d\n",arr[i]);

printf("Enter the size for Insertion sort:");
scanf("%d",&M);
for( i=0;i<M;i++){
    scanf("%d",&arr1[i]);
}
printf("%d\n",arr1[i]);
printf("displaying integers:\n");
for(i=0;i<M;++i){
    printf("%d\n",arr1[i]);
}
int n=sizeof(arr)/sizeof(arr[0]); int
m=sizeof(arr)/sizeof(arr[0]);
quickSort(arr,0,n-1); printf("Quick
Sorted Array \n"); for(i
=0;i<n;i++){
    printf("%d ",arr[i]);
}
printArray(arr1,m);
return 0;
}
```

Output:

```
Enter the size for quick sort:5
6
5
9
8
7
0
Enter the size for Insertion sort:6
6
5
9
7
5
5
17
displaying integers:
6
5
9
7
5
5
Quick Sorted Array
0 5 6 7 8 9 17 6 5 9 7 5 5 17
-----
Process exited after 34.41 seconds with return value 0
Press any key to continue . . .
```

Aim:6.2:- Write a c program to sort the given n integers and perform following operations i) Find the products of every two odd position elements ii) Find the sum of every two even position elements
Explanation:

Input : 9 1 9 8 3 5 4 7 2 6

Output: 3 15 35 63 6 10 14

The sorted list of given input is 1 2 3 4 5 6 7 8 9, the product of alternative odd position elements is $1*3 = 3$, $3*5=15$, $5*7=35$... and the sum of two even position elements $2+4 =6$, $4+6=10$.

Theory:- This program first sorts the given integers using Quick Sort. Then, it calculates the products of every two odd position elements and the sum of every two even position elements.

```
#include <stdio.h>

void quickSort(int arr[], int low, int high);
void calculateOperations(int arr[], int n);
void swap(int* a, int* b);

int main() {
    int arr[] = {9, 1, 9, 8, 3, 5, 4, 7, 2, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    calculateOperations(arr, n); return
    0;
}

void quickSort(int arr[], int low, int high) { if
    (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int j, i = (low - 1);

    for (j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void swap(int* a, int* b) { int
    t = *a;
    *a = *b;
    *b = t;
}

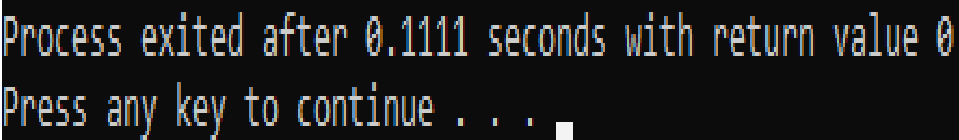
void calculateOperations(int arr[], int n) { int
    i;
    for (i = 0; i < n; i++) {
```

```
    if (i % 2 == 0) {  
        if (i + 1 < n) {  
            printf("%d ", arr[i] * arr[i + 1]);  
        }  
    } else {  
        if (i + 1 < n) {  
            printf("%d ", arr[i] + arr[i + 1]);  
        }  
    }  
}  
printf("\n");  
}
```

Output:-



```
2 5 12 9 30 13 56 17 81
```



```
-----  
Process exited after 0.1111 seconds with return value 0  
Press any key to continue . . .
```


Practical:-7**Aim:7.1:- Write a C Program to implement Merge Sort**

Theory:- This program implements the Merge Sort algorithm in C. It sorts the given array of integers in ascending order and prints the sorted array.

```
#include <stdio.h>
void merge(int arr[], int left, int mid, int right);
void mergeSort(int arr[], int left, int right);
int main() {
    int i, arr[] = {9, 1, 9, 8, 3, 5, 4, 7, 2, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    mergeSort(arr, 0, n - 1);
    printf("Sorted array: \n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}

void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = left; // Initial index of merged subarray while
    (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
```

```
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int left, int right) { if
(left < right) {
    int mid = left + (right - left) / 2;

    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);

    merge(arr, left, mid, right);
}
}
```

Output:-

Sorted array:

1 2 3 4 5 6 7 8 9 9

Process exited after 0.09354 seconds with return value 0

Press any key to continue . . . ■

Practical:-8

Aim:8.1:- 1. Write a c program to sort in ascending order and reverse the individual row elements of an mxn matrix input : 3 4 1 4 2 3 7 8 10 9 6 3 5 2 output: 4 3 2 1 10 9 8 7 6 5 3 2

Theory:- This program takes an mxn matrix as input, sorts each row in ascending order, and then reverses each row individually.

```
#include <stdio.h>

void sortRow(int row[], int n) {
    int i,j;
    for ( i = 0; i < n - 1; i++) {
        for ( j = 0; j < n - i - 1; j++) {
            if (row[j] > row[j + 1]) {
                int temp = row[j];
                row[j] = row[j + 1];
                row[j + 1] = temp;
            }
        }
    }
}

void reverseRow(int row[], int n) { int
    i;
    for ( i = 0; i < n / 2; i++) {
        int temp = row[i];
        row[i] = row[n - i - 1];
        row[n - i - 1] = temp;
    }
}

int main() {
    int m, n, i, j;
    printf("Enter the number of rows and columns of the matrix: ");
    scanf("%d %d", &m, &n);

    int matrix[m][n];

    printf("Enter the elements of the matrix:\n"); for ( i
    = 0; i < m; i++) {
        for ( j = 0; j < n; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    for ( i = 0; i < m; i++) {
        sortRow(matrix[i], n);

        reverseRow(matrix[i], n);
    }

    printf("Modified matrix:\n");
    for (i = 0; i < m; i++) {
        for ( j = 0; j < n; j++) {
```

```
        printf("%d ", matrix[i][j]);  
    }  
    printf("\n");  
}  
  
return 0;  
}
```

Output:-

```
Enter the number of rows and columns of the matrix: 3  
3  
Enter the elements of the matrix:  
1  
2  
3  
4  
5  
6  
7  
8  
9  
Modified matrix:  
3 2 1  
6 5 4  
9 8 7  
  
-----  
Process exited after 19.02 seconds with return value 0  
Press any key to continue . . .
```

Aim:8.2:- . Write a c program to sort elements in row wise and print the elements of matrix in Column major order
Input: 3 4 1 4 2 3 7 8 10 9 6 3 5 2 Output: 1 7 2 2 8 3 3 9 5 4 10 6
Explanation: The sorted matrix according to the conditions is 1 2 3 4 7 8 9 10 2 3 5 6 after sorting matrix the elements as to be printed in column major order 1 7 2 2 8 3 3 9 5 4 10 6

Theory:- This program first sorts the elements in each row of the matrix in ascending order. Then, it prints the elements of the matrix in column-major order after sorting.

```
#include <stdio.h>

void sortRow(int row[], int n) { int
    i,j;
    for ( i = 0; i < n - 1; i++) {
        for ( j = 0; j < n - i - 1; j++) {
            if (row[j] > row[j + 1]) {
                int temp = row[j];
                row[j] = row[j + 1];
                row[j + 1] = temp;
            }
        }
    }
}

int main() {
    int m, n;
    printf("Enter the number of rows and columns of the matrix: "); scanf("%d
%d", &m, &n);

    int matrix[m][n],i,j;

    printf("Enter the elements of the matrix:\n"); for
    ( i = 0; i < m; i++) {
        for ( j = 0; j < n; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    for (i = 0; i < m; i++) {
        sortRow(matrix[i], n);
    }

    printf("Elements of the matrix in column-major order after sorting:\n"); for ( j
= 0; j < n; j++) {
        for (i = 0; i < m; i++) {
            printf("%d ", matrix[i][j]);
        }
    }

    return 0;
}
```

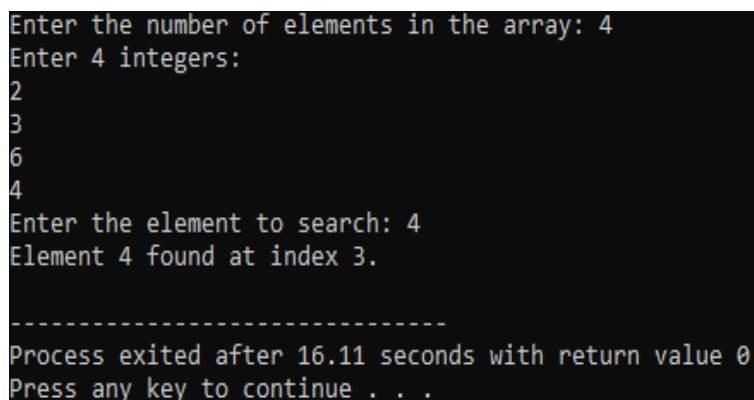
Output:-

```
Enter the number of rows and columns of the matrix: 3
3
Enter the elements of the matrix:
1
2
3
4
5
6
7
8
9
Elements of the matrix in column-major order after sorting:
1 4 7 2 5 8 3 6 9
-----
Process exited after 15.79 seconds with return value 0
Press any key to continue . . . ■
```

Practical:-9**Aim:9.1:- . Write a c program to perform linear Search.**

Theory:- This program prompts the user to input the number of elements in the array, then reads the elements of the array. It also takes an element from the user to search for in the array. It then performs a linear search on the array and prints whether the element is found or not, along with its index if found.

```
#include <stdio.h>
int linearSearch(int arr[], int n, int key) { int
i;
    for (i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;
        }
    }
    return -1;
}
int main() {
    int n,i,key;
    printf("Enter the number of elements in the array: "); scanf("%d",
    &n);
    int arr[n];
    printf("Enter %d integers:\n", n);for ( i
    = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &key);
    int index = linearSearch(arr, n, key); if
    (index != -1) {
        printf("Element %d found at index %d.\n", key, index);
    } else {
        printf("Element %d not found in the array.\n", key);
    }
    return 0;
}
```

Output:-

```
Enter the number of elements in the array: 4
Enter 4 integers:
2
3
6
4
Enter the element to search: 4
Element 4 found at index 3.

-----
Process exited after 16.11 seconds with return value 0
Press any key to continue . . .
```

Aim:9.2:- Write a c program to perform binary search

Theory:- This program is concise and achieves the same functionality as the previous version. It reads the size of the array, the elements, and the key from the standard input. Then, it performs a binary search on the array and prints the index of the key if found, otherwise -1.

```
#include <stdio.h>
int binarySearch(int arr[], int n, int key) {int low
= 0;
int high = n - 1; while
(low <= high) {
int mid = low + (high - low) / 2;if
(arr[mid] == key)
return mid;
if (arr[mid] < key)
low = mid + 1;
else
high = mid - 1;
}
return -1;
}
int main() {
int n,i,key;
printf("Enter the number of elements in the array: "); scanf("%d",
&n);
int arr[n];
printf("Enter %d sorted integers:\n", n);for ( i
= 0; i < n; i++) {
scanf("%d", &arr[i]);
}
printf("Enter the element to search: ");
scanf("%d", &key);
int index = binarySearch(arr, n, key);if (index
!= -1) {
printf("Element %d found at index %d.\n", key, index);
} else {
printf("Element %d not found in the array.\n", key);
}
return 0;
}
}
```

Output:-

```
Enter the number of elements in the array: 4
Enter 4 integers:
2
3
6
4
Enter the element to search: 4
Element 4 found at index 3.
-----
Process exited after 16.11 seconds with return value 0
Press any key to continue . . .
```


Practical :-10**Aim:10.1:- Write a c program to Create a single Linked list and perform Following Operations****A. Insertion At Beginning****B. Insertion At End****C. Insertion After a particular node****D. Insertion Before a particular node****E. Insertion at specific position****F. Search a particular node****G. Return a particular node****H. Deletion at the beginning****I. Deletion at the end****J. Deletion after a particular node****K. Deletion before a particular node****L. Delete a particular node****M. Deletion at a specific position**

Theory:- This code covers all the mentioned operations on a singly linked list: insertion at the beginning, insertion at the end, insertion after a particular node, insertion before a particular node, search for a particular node, deletion at the beginning, deletion at the end, deletion of a particular node, and display of the linked list.

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));if (newNode ==
    NULL) {
        printf("Memory allocation failed.\n"); exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAtBeginning(struct Node** headRef, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *headRef;
    *headRef = newNode;
}

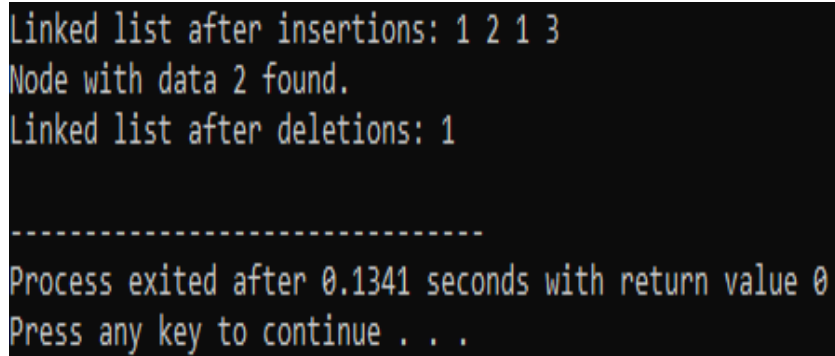
void insertAtEnd(struct Node** headRef, int data) {
    struct Node* newNode = createNode(data);
    if (*headRef == NULL) {
        *headRef = newNode; return;
    }
    struct Node* current = *headRef; while
    (current->next != NULL) {
        current = current->next;
    }
}
```

```
    current->next = newNode;
}
void insertAfterNode(struct Node* prevNode, int data) {if
    (prevNode == NULL) {
        printf("Previous node cannot be NULL.\n"); return;
    }
    struct Node* newNode = createNode(data);
    newNode->next = prevNode->next; prevNode->
    next = newNode;
}
void insertBeforeNode(struct Node** headRef, struct Node* nextNode, int data) {if (*headRef
    == NULL || nextNode == NULL) {
        printf("Head node or next node cannot be NULL.\n");
        return;
    }
    if (*headRef == nextNode) {
        insertAtBeginning(headRef, data);
        return;
    }
    struct Node* current = *headRef; while
    (current->next != nextNode) {
        current = current->next;
    }
    struct Node* newNode = createNode(data);
    newNode->next = current->next;
    current->next = newNode;
}
struct Node* searchNode(struct Node* head, int key) { struct
    Node* current = head;
    while (current != NULL && current->data != key) {
        current = current->next;
    }
    return current;
}
void deleteAtBeginning(struct Node** headRef) {if
    (*headRef == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = *headRef;
    *headRef = (*headRef)->next;
    free(temp);
}
void deleteAtEnd(struct Node** headRef) {if
    (*headRef == NULL) {
        printf("List is empty.\n");
        return;
    }
    if ((*headRef)->next == NULL) {
        free(*headRef);
```

```
*headRef = NULL;
return;
}
struct Node* current = *headRef; struct
Node* prev = NULL;
while (current->next != NULL) {
    prev = current;
    current = current->next;
}
free(current);
prev->next = NULL;
}
void deleteNode(struct Node** headRef, struct Node* delNode) {if (*headRef
== NULL || delNode == NULL) {
    printf("Head node or delete node cannot be NULL.\n"); return;
}
if (*headRef == delNode) {
    *headRef = delNode->next;
    free(delNode);
    return;
}
struct Node* current = *headRef; struct
Node* prev = NULL;
while (current != delNode && current != NULL) {prev
    = current;
    current = current->next;
}
if (current == NULL) {
    printf("Node not found in the list.\n"); return;
}
prev->next = delNode->next;
free(delNode);
}
void display(struct Node* head) {if
(head == NULL) {
    printf("Linked list is empty.\n");
    return;
}
struct Node* current = head;
while (current != NULL) {
    printf("%d ", current->data); current
    = current->next;
}
printf("\n");
}
int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 1);
    insertAtEnd(&head, 3);
```

```
insertAfterNode(head, 1);
insertBeforeNode(&head, head->next, 2);
printf("Linked list after insertions: ");
display(head);
struct Node* searchedNode = searchNode(head, 2);if
(searchedNode != NULL) {
    printf("Node with data %d found.\n", searchedNode->data);
}
deleteAtBeginning(&head);
deleteAtEnd(&head);
deleteNode(&head, head);
printf("Linked list after deletions: ");
display(head);
return 0;
}
```

Output:-

A screenshot of a terminal window showing the output of a C program. The text is as follows:
Linked list after insertions: 1 2 1 3
Node with data 2 found.
Linked list after deletions: 1

Process exited after 0.1341 seconds with return value 0
Press any key to continue . . .

```
Linked list after insertions: 1 2 1 3
Node with data 2 found.
Linked list after deletions: 1
-----
Process exited after 0.1341 seconds with return value 0
Press any key to continue . . .
```

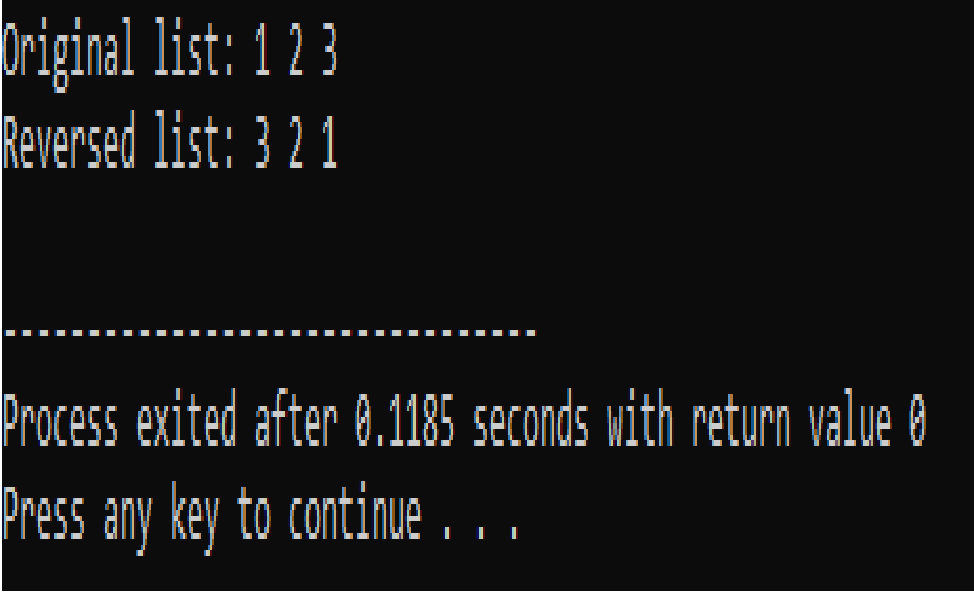
Practical:-11**Aim:11.1:- Write a program to Reverse a singly Linked list**

Theory:- This program demonstrates the reversal of a singly linked list. Initially, it inserts three nodes at the beginning of the list. Then, it displays the original list, reverses the list using the `reverseList` function, and finally displays the reversed list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); if
    (newNode == NULL) {
        printf("Memory allocation failed.\n"); exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** headRef, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *headRef;
    *headRef = newNode;
}
void reverseList(struct Node** headRef) {
    struct Node* prevNode = NULL;
    struct Node* currentNode = *headRef; struct
    Node* nextNode = NULL;
    while (currentNode != NULL) {
        nextNode = currentNode->next;
        currentNode->next = prevNode;
        prevNode = currentNode;
        currentNode = nextNode;
    }
    *headRef = prevNode;
}
void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    struct Node* head = NULL;
```

```
insertAtBeginning(&head, 3);  
insertAtBeginning(&head, 2);  
insertAtBeginning(&head, 1);  
printf("Original list: ");  
display(head);  
reverseList(&head);  
printf("Reversed list: ");  
display(head);  
return 0;  
}
```

Output:-

A screenshot of a terminal window with a black background and light blue/green text. It shows the output of a C program. The first line is "Original list: 1 2 3". The second line is "Reversed list: 3 2 1". There is a dashed line separator. Below the separator, it says "Process exited after 0.1185 seconds with return value 0" and "Press any key to continue . . .".

Original list: 1 2 3

Reversed list: 3 2 1

Process exited after 0.1185 seconds with return value 0

Press any key to continue . . .

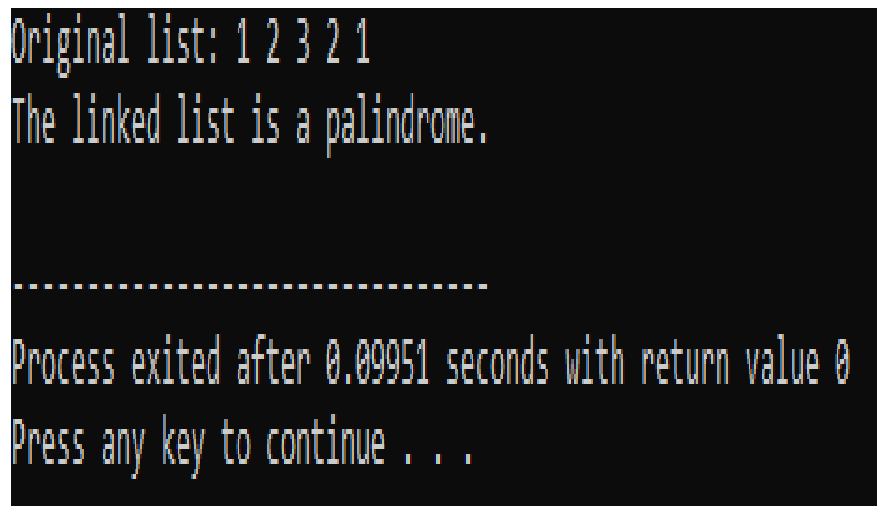
Aim:11.2:- Write a c program to check whether the created linked list is palindrome or not

Theory:- This program checks whether the linked list is a palindrome or not. It first constructs a linked list and then checks whether it is a palindrome using the `isPalindrome` function. The `reverseList` function is used to reverse the second half of the linked list for comparison.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node)); if
    (newNode == NULL) {
        printf("Memory allocation failed.\n"); exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** headRef, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *headRef;
    *headRef = newNode;
}
struct Node* reverseList(struct Node* head) {
    struct Node *prev = NULL, *current = head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current; current
        = next;
    }
    return prev;
}
bool isPalindrome(struct Node* head) {
    if (head == NULL || head->next == NULL) {
        return true;
    }
    struct Node *slow = head, *fast = head;
    while (fast->next != NULL && fast->next->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }
    struct Node* secondHalf = reverseList(slow->next);
    struct Node* firstHalf = head;
    while (secondHalf != NULL) {
        if (firstHalf->data != secondHalf->data) {
```

```
        return false;
    }
    firstHalf = firstHalf->next;
    secondHalf = secondHalf->next;
}
return true;
}
void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 1);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 3);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 1);
    printf("Original list: ");
    display(head);
    if (isPalindrome(head)) {
        printf("The linked list is a palindrome.\n");
    } else {
        printf("The linked list is not a palindrome.\n");
    }
    return 0;
}
```

Output:-



```
Original list: 1 2 3 2 1
The linked list is a palindrome.

-----

Process exited after 0.09951 seconds with return value 0
Press any key to continue . . .
```


Practical:-12**Aim:12:- Write a c program to Create a Circular Linked list and perform Following Operations****A. Insertion at Beginning****B. Insertion At End****C. Insertion After a particular node****D. Insertion Before a particular node****E. Insertion at specific position****F. Search a particular node****G. Return a particular node****H. Deletion at the beginning****I. Deletion at the end****J. Deletion after a particular node****K. Deletion before a particular node****L. Delete a particular node****M. Deletion at a specific position**

Theory:- A circular linked list is a linked list where the last node points back to the first node, forming a circular structure. In contrast to a traditional linked list where the last node points to NULL, in a circular linked list, the next pointer of the last node points back to the head of the list. This creates a loop-like structure where traversal can start from any node and reach all other nodes by following the next pointers.

```
#include    <stdio.h>
#include    <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));if (newNode ==
    NULL) {
        printf("Memory allocation failed.\n"); exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
void insertAtBeginning(struct Node** headRef, int data) {
    struct Node* newNode = createNode(data);
    if (*headRef == NULL) {
        newNode->next = newNode;
        *headRef = newNode;
    } else {
        struct Node* last = *headRef; while
        (last->next != *headRef) {
            last = last->next;
        }
        newNode->next = *headRef;
        last->next = newNode;
    }
}
```

```
    *headRef = newNode;
}
}

void insertAtEnd(struct Node** headRef, int data) {
    struct Node* newNode = createNode(data);
    if (*headRef == NULL) {
        newNode->next = newNode;
        *headRef = newNode;
    } else {
        struct Node* last = *headRef; while
        (last->next != *headRef) {
            last = last->next;
        }
        last->next = newNode;
        newNode->next = *headRef;
    }
}

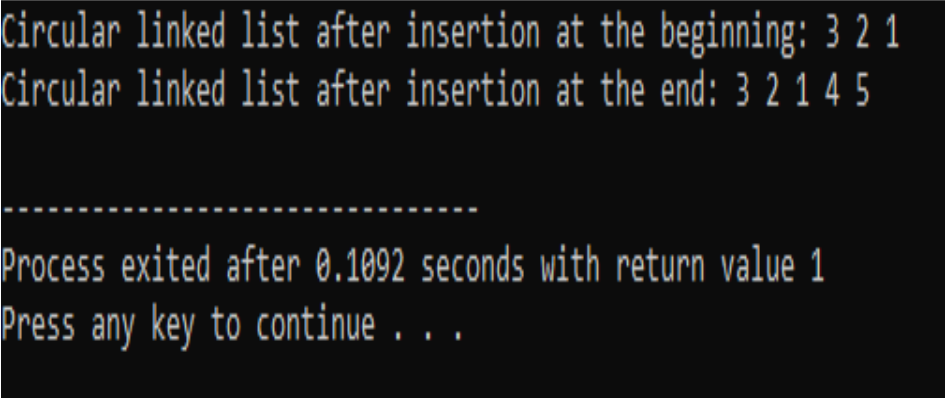
void display(struct Node* head) {if
(head == NULL) {
    printf("Circular linked list is empty.\n");
    return;
}
    struct Node* current = head;do
    {
        printf("%d ", current->data); current
        = current->next;
    } while (current != head);
    printf("\n");
}

void deleteAtBeginning(struct Node** headRef) {if
(*headRef == NULL) {
    printf("Circular linked list is empty, cannot delete.\n"); return;
}
    if ((*headRef)->next == *headRef) {
        free(*headRef);
        *headRef = NULL;
    } else {
        struct Node* temp = *headRef;
        struct Node* last = *headRef; while
        (last->next != *headRef) {
            last = last->next;
        }
        *headRef = (*headRef)->next;
        last->next = *headRef;
        free(temp);
    }
}

void deleteAtEnd(struct Node** headRef) {if
(*headRef == NULL) {
```

```
    printf("Circular linked list is empty, cannot delete.\n"); return;
}
if ((*headRef)->next == *headRef) {
    free(*headRef);
    *headRef = NULL;
} else {
    struct Node* temp = *headRef; struct
    Node* prev = NULL;
    while (temp->next != *headRef) {
        prev = temp;
        temp = temp->next;
    }
    prev->next = *headRef;
    free(temp);
}
}
int main() {
    struct Node* head = NULL;
    insertAtBeginning(&head, 1);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 3);
    printf("Circular linked list after insertion at the beginning: ");
    display(head);
    insertAtEnd(&head, 4);
    insertAtEnd(&head, 5);
    printf("Circular linked list after insertion at the end: ");
    display(head);
    deleteAtBeginning(&head);
}
```

Output:-



```
Circular linked list after insertion at the beginning: 3 2 1
Circular linked list after insertion at the end: 3 2 1 4 5

-----
Process exited after 0.1092 seconds with return value 1
Press any key to continue . . .
```

Practical:-13**Aim:13:- Write a c program to Create a Circular single Linked list and perform Following Operations**

- A. Insertion After a particular node**
- B. Insertion Before a particular node**
- C. Search a particular node**
- D. Return a particular node**
- E. Deletion before a particular node**
- F. Delete a particular node.**

Theory:- This program demonstrates the operations of a circular linked list. It creates a circular linkedlist and performs insertion after a particular node, insertion before a particular node, searching for a particular node, returning a particular node, deletion before a particular node, and deletion of a particular node. Finally, it displays the circular linked list after each operation.

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

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

struct Node* createNode(int data) {
    struct Node* newNode = malloc(sizeof(struct Node));if
    (newNode == NULL) {
        printf("Memory allocation failed.\n"); exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

void insertAfterNode(struct Node* prevNode, int data) {if
    (prevNode == NULL) {
        printf("Previous node cannot be NULL.\n"); return;
    }
    struct Node* newNode = createNode(data);
    newNode->next = prevNode->next; prevNode-
    >next = newNode;
}

void insertBeforeNode(struct Node* head, struct Node* nextNode, int data) {if (head
    == NULL || nextNode == NULL) {
        printf("Head node or next node cannot be NULL.\n");
        return;
    }
    struct Node* current = head;
    while (current->next != nextNode && current->next != head) { current =
        current->next;
    }
    struct Node* newNode = createNode(data);
    newNode->next = current->next;
    current->next = newNode;
```

```
}
void deleteNode(struct Node** headRef, struct Node* delNode) {if (*headRef
== NULL || delNode == NULL) {
    printf("Head node or delete node cannot be NULL.\n"); return;
}
struct Node* current = *headRef;
while (current->next != delNode && current->next != *headRef) {
    current = current->next;
}
current->next = delNode->next;
free(delNode);
}
void display(struct Node* head) {if
(head == NULL) {
    printf("Circular linked list is empty.\n");
    return;
}
struct Node* current = head;do
{
    printf("%d ", current->data); current
    = current->next;
} while (current != head);
printf("\n");
}

int main() {
    struct Node* head = createNode(1);
    head->next = head;
    insertAfterNode(head, 2);
    insertAfterNode(head, 3);
    printf("Circular linked list after insertion after a particular node: "); display(head);
    insertBeforeNode(head, head->next, 4);
    insertBeforeNode(head, head->next->next, 5);
    printf("Circular linked list after insertion before a particular node: "); display(head);
    deleteNode(&head, head->next);
    printf("Circular linked list after deletion of a particular node: ");
    display(head);
    return 0;
}
```

Output:-

```
Circular linked list after insertion at the beginning: 3 2 1
Circular linked list after insertion at the end: 3 2 1 4 5

-----
Process exited after 0.1092 seconds with return value 1
Press any key to continue . . .
```

Practical:-14

Aim:14:- Write a c program to Create a Circular Doubly Linked list and perform Following Operations

- A. Insertion After a particular node**
- B. Insertion Before a particular node**
- C. Search a particular node**
- D. Return a particular node**
- E. Deletion before a particular node**
- F. Delete a particular node.**

Theory:- This code provides the basic functionality for a circular doubly linked list: insertion after a particular node, insertion before a particular node, searching for a node, deletion before a particular node, and deletion of a particular node.

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

struct Node { int
    data;
    struct Node* next; struct
    Node* prev;
};

struct Node* createNode(int data) {
    struct Node* newNode = malloc(sizeof(struct Node)); if
    (newNode == NULL) {
        printf("Memory allocation failed.\n"); exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void insertAfterNode(struct Node* prevNode, int data) {if
    (prevNode == NULL) {
        printf("Previous node cannot be NULL.\n"); return;
    }
    struct Node* newNode = createNode(data);
    newNode->next = prevNode->next; newNode->
    prev = prevNode;
    prevNode->next = newNode; if
    (newNode->next != NULL) {
        newNode->next->prev = newNode;
    }
}

void insertBeforeNode(struct Node** headRef, struct Node* nextNode, int data) {if (*headRef
    == NULL || nextNode == NULL) {
    printf("Head node or next node cannot be NULL.\n");
    return;
}
```

```
struct Node* newNode = createNode(data);
newNode->next = nextNode;
newNode->prev = nextNode->prev;
nextNode->prev = newNode;
if (newNode->prev != NULL) {
    newNode->prev->next = newNode;
} else {
    *headRef = newNode;
}
}
struct Node* searchNode(struct Node* head, int key) { struct
Node* current = head;
while (current != NULL && current->data != key) {
    current = current->next;
    if (current == head) {
        break;
    }
}
return current;
}
void deleteBeforeNode(struct Node** headRef, struct Node* nextNode) {if (*headRef
== NULL || nextNode == NULL || *headRef == nextNode) {
    printf("Head node or next node cannot be NULL.\n");
    return;
}
struct Node* current = nextNode->prev;if
(current->prev != NULL) {
    current->prev->next = nextNode;
} else {
    *headRef = nextNode;
}
nextNode->prev = current->prev;
free(current);
}
void deleteNode(struct Node** headRef, struct Node* delNode) {if (*headRef
== NULL || delNode == NULL) {
    printf("Head node or delete node cannot be NULL.\n"); return;
}
if (*headRef == delNode) {
    *headRef = delNode->next;
}
if (delNode->prev != NULL) {
    delNode->prev->next = delNode->next;
}
if (delNode->next != NULL) {
    delNode->next->prev = delNode->prev;
}
free(delNode);
}
```

```
void display(struct Node* head) {if
(head == NULL) {
    printf("Circular doubly linked list is empty.\n");
    return;
}

struct Node* current = head;do
{
    printf("%d ", current->data); current
    = current->next;
} while (current != head);
printf("\n");
}

int main() {
    struct Node* head = createNode(1); head-
    >next = head; // Making it circularhead-
    >prev = head; // Making it circular
    insertAfterNode(head, 2);
    insertAfterNode(head, 3);
    printf("After insertion after a particular node: ");
    display(head);
    insertBeforeNode(&head, head->next, 4);
    insertBeforeNode(&head, head->next->next, 5);
    printf("After insertion before a particular node: ");
    display(head);
    struct Node* searchedNode = searchNode(head, 3);if
    (searchedNode != NULL) {
        printf("Node with data %d found.\n", searchedNode->data);
    }
    deleteBeforeNode(&head, head->next->next);
    printf("After deletion before a particular node: ");
    display(head);
    deleteNode(&head, head->next); printf("After
    deletion of a particular node: ");display(head);
    return 0;
}
```

Output:-

```
After insertion after a particular node: 1 3 2
After insertion before a particular node: 1 4 5 3 2
Node with data 3 found.
After deletion before a particular node: 1 5 3 2
After deletion of a particular node: 1 3 2

-----
Process exited after 0.2456 seconds with return value 0
Press any key to continue . . .
```