1. **Input / Output** of Array
2. **Minimum & Maximum element**
3. **Second Largest element**
4. **Reverse Array**
5. **Sum and Average**
6. **Even and Odd count**
7. **Linear Search**
8. **Binary Search**
9. **Frequency of Elements**
10. **Move Zeroes to End**
11. **Find Duplicates**
12. **Left Rotation by k**
13. **Right Rotation by k**
14. **Sort 0s, 1s, 2s** (Dutch National Flag)
15. **Missing Number** (from 1 to n)
16. **Dynamic Memory Allocation** using

- malloc
- calloc
- realloc
- free

```
//#include<stdlib.h>
//#include<stdio.h>
//int main(){
//	int n;
//	printf("Enter the size of array:");
//	scanf("%d",&n);
//	int a[n],i;
//	printf("Enter the elements of an array:\n");
//	for(i=0;i<n;i++){
//		printf("arr[%d]=",i);
//		scanf("%d",&a[i]);
//	}
//	printf("a[%d]=\t",n);
//	for(i=0;i<n;i++){
//		printf("%d\t",a[i]);
//	}

	//Minimum and maximum element of array

//	int min,max;
//	min=a[0];
//	max=a[0];
//	for(i=0;i<n;i++){
//		if(a[i]<min){
//			min=a[i];
//		}
//		else if(a[i]>max){
//			max=a[i];
//		}
//	}
//	printf("\nMin element=%d\t max element=%d\n",min,max);

	//2nd largest element of an array
//	int first=-1;
//	int second=-1;
//	for(i=0;i<n;i++){
//		if(a[i]>first){
//			second=first;
//			first=a[i];
//		}
//		else if(a[i]>second && a[i]<first){
//			second=a[i];
```

```
//                  }
//          }
//          printf("2nd largest is=%d\n",second);

            //reverse array
//          int start=0;
//          int end=n-1;
//          int temp;
//          while(start<end){
//                  temp=a[start];
//                  a[start]=a[end];
//                  a[end]=temp;
//                  start++;
//                  end--;
//          }
//                  printf("a[%d]=\t",n);
//          for(i=0;i<n;i++){
//                  printf("%d\t",a[i]);
//          }

            //Sum and avg of array
//          int sum=0;
//          float avg=0;
//          for(i=0;i<n;i++){
//                  sum=sum+a[i];
//          }
//          avg=(float)sum/n;
//          printf("sum =%d\t avg=%.2f\n",sum,avg);

            //Even and odd elements of array
//          int even=0;
//          int odd=0;
//          for(i=0;i<n;i++){
//                  if(a[i]%2==0){
//                          even++;
//                  }
//                  else{
//                          odd++;
//                  }
//          }
//          printf("even=%d\t odd=%d\n",even,odd);

            //Linear search
```

```
//	int search,flag=0;
//	printf("Enter the element you want to search:");
//	scanf("%d",&search);
//	for(i=0;i<n;i++){
//		if(a[i]==search){
//			flag=1;
//			break;
//		}
//	}
//	if(flag==1){
//	printf("%d element found at %d position\n",search,i+1);
//	}
//	if(flag==0){
//		printf("Element not found!");
//	}

	//Binary search
//	int search,flag=0;
//	printf("Enter the element you want to search in sorted array:");
//	scanf("%d",&search);
//	int mid,start=0,end=n-1;
//	while(start<=end){
//		mid=(start+end)/2;
//		if(a[mid]==search){
//			flag=1;
//			break;
//		}
//		else if(a[mid]>search){
//			end=mid-1;
//		}
//		else{
//			start=mid+1;
//		}
//	}
//	if(flag==1){
//	printf("%d element found at %d position\n",search,mid+1);
//	}
//	if(flag==0){
//		printf("Element not found!");
//	}


	//Frequency of each element
```

```
//	int visit[n];
//	int j,count=0;
//	for(i=0;i<n;i++){
//		if(visit[i]==1){
//			continue;
//		}
//		else{
//			count=1;
//			for(j=i+1;j<n;j++){
//				if(a[i]==a[j]){
//					count++;
//					visit[j]=1;
//				}
//			}
//			printf("%d element occurs %d times\n",a[i],count);
//		}
//	}

	//move all zeroes at the end
//	int count=0;
//	for(i=0;i<n;i++){
//		if(a[i]!=0){
//			a[count]=a[i];
//			count++;
//		}
//	}
//	while(count<n){
//		a[count]=0;
//		count++;
//	}
//		printf("New array a[%d]=\t",n);
//	for(i=0;i<n;i++){
//		printf("%d\t",a[i]);
//	}

	//duplicate elements
//	int j;
//	for(i=0;i<n;i++){
//		for(j=i+1;j<n;j++){
//			if(a[i]==a[j]){
//				printf("%d is duplicate\n",a[i]);
//				break;
//			}
```

```
//                  }
//          }

            //K position shifting of array(left)
//          int left,start,j;
//          printf("Enter position to left shift:");
//          scanf("%d",&left);
//          for(i=0;i<left;i++){
//                  start=a[0];
//                  for(j=0;j<n-1;j++){
//                          a[j]=a[j+1];
//                  }
//                  a[n-1]=start;
//          }
//          printf("New array a[%d]=\t",n);
//          for(i=0;i<n;i++){
//                  printf("%d\t",a[i]);
//          }

            //K position shifting of array(right)
//          int right,end,j;
//          printf("Enter position to right shift:");
//          scanf("%d",&right);
//          for(i=0;i<right;i++){
//                  end=a[n-1];
//                  for(j=n-1;j>0;j--){
//                          a[j]=a[j-1];
//                  }
//                  a[0]=end;
//          }
//          printf("New array a[%d]=\t",n);
//          for(i=0;i<n;i++){
//                  printf("%d\t",a[i]);
//          }

            //Sort 0,1,2
//          int low = 0, mid = 0, high = n - 1;
//    while(mid <= high) {
//       if(a[mid] == 0) {
//          int temp = a[low];
//          a[low] = a[mid];
//          a[mid] = temp;
//          low++;
```

```
//         mid++;
//      }
//      else if(a[mid] == 1) {
//         mid++;
//      }
//      else {
//         int temp = a[mid];
//         a[mid] = a[high];
//         a[high] = temp;
//         high--;
//      }
//   }
//   printf("Array after sorting 0s,1s,2s: ");
//   for(i = 0; i < n; i++) {
//      printf("%d ", a[i]);
//   }

         //Missing no.
//         int expected_sum = n * (n + 1) / 2;
//   int actual_sum = 0;
//   for(i = 0; i < n-1; i++) {
//      actual_sum += a[i];
//   }
//   int missing = expected_sum - actual_sum;
//   printf("Missing number is: %d\n", missing);


         //Malloc,Calloc and Realloc
//         int *arr,i;
//   arr = (int*) malloc(n * sizeof(int));
//   if (arr == NULL) {
//      printf("Memory not allocated!\n");
//      return 1;
//   }
//   printf("\nEnter %d elements:\n", n);
//   for (i = 0; i < n; i++) {
//      scanf("%d", &arr[i]);
//   }
//   printf("Array elements are:\n");
//   for (i = 0; i < n; i++) {
//      printf("%d ", arr[i]);
//   }
//   printf("\n\nUsing calloc...\n");
```

```
//   arr = (int*) calloc(n, sizeof(int));
//   if (arr == NULL) {
//      printf("Memory not allocated!\n");
//      return 1;
//   }
//   printf("Array initialized with calloc:\n");
//   for (i = 0; i < n; i++) {
//      printf("%d ", arr[i]);
//   }
//   printf("\n\nEnter new size for array: ");
//   scanf("%d", &n);
//   arr = (int*) realloc(arr, n * sizeof(int));
//   if (arr == NULL) {
//      printf("Memory not reallocated!\n");
//      return 1;
//   }
//   printf("Enter %d elements:\n", n);
//   for (i = 0; i < n; i++) {
//      scanf("%d", &arr[i]);
//   }
//   printf("New array elements are:\n");
//   for (i = 0; i < n; i++) {
//      printf("%d ", arr[i]);
//   }
//   free(arr);
//}
```

```
#include<stdio.h>
int main(){
//	int m,n,i,j;
//	printf("Enter no. of rows:");
//	scanf("%d",&m);
//	printf("Enter no. of columns:");
//	scanf("%d",&n);
//	int a[m][n];
//	printf("Enter elements in matrix:\n");
//	for(i=0;i<m;i++){
//		for(j=0;j<n;j++){
//			printf("a[%d][%d]=",i,j);
//			scanf("%d",&a[i][j]);
//		}
//	}
//	printf("Matrix A:\n");
//	for(i=0;i<m;i++){
//		for(j=0;j<n;j++){
//			printf("%d\t",a[i][j]);
//		}
//		printf("\n");
//	}

	//row sum
//	int row;
//		for(i=0;i<m;i++){
//			row=0;
//		for(j=0;j<n;j++){
//			row=row+a[i][j];
```

```
//                  }
//                  printf("Sum of row %d:%d\n",i+1,row);

        //column sum
//      int column;
//                  for(i=0;i<m;i++){
//                          column=0;
//                  for(j=0;j<n;j++){
//                          column=column+a[j][i];
//                  }
//                  printf("Sum of column %d:%d\n",i+1,column);
//      }

        //Transpose
//      printf(" Transpose Matrix :\n");
//      for(i=0;i<m;i++){
//                  for(j=0;j<n;j++){
//                          printf("%d\t",a[j][i]);
//                  }
//                  printf("\n");
//      }

        //diagonal sum
//      int diagonal;
//      for(i=0;i<m;i++){
//                  for(j=0;j<n;j++){
//                          if(i==j){
//                                  diagonal=diagonal+a[i][j];
//                          }
```

```
//                  }
//          }
//          printf("Diagonal sum=%d\n",diagonal);

            //Upper triangular matrix
//          for(i=0;i<m;i++){
//                  for(j=0;j<n;j++){
//                          if(i>j){
//                                  a[i][j]=0;
//                          }
//                          printf("%d\t",a[i][j]);
//                  }
//                  printf("\n");
//          }

            //Lower triangluar matrix
//                  for(i=0;i<m;i++){
//                  for(j=0;j<n;j++){
//                          if(j>i){
//                                  a[i][j]=0;
//                          }
//                          printf("%d\t",a[i][j]);
//                  }
//                  printf("\n");
//          }

            //2D matrix
            int m,n,p,q,i,j;
            printf("Enter elements of matrix 1:");
```

```
	printf("Enter no. of rows:");
	scanf("%d",&m);
	printf("Enter no. of columns:");
	scanf("%d",&n);
	int a[m][n];
	printf("Enter elements in matrix:\n");
	for(i=0;i<m;i++){
		for(j=0;j<n;j++){
			printf("a[%d][%d]=",i,j);
			scanf("%d",&a[i][j]);
		}
	}
	printf("Enter elements of matrix 2:");
	printf("Enter no. of rows:");
	scanf("%d",&p);
	printf("Enter no. of columns:");
	scanf("%d",&q);
	int b[p][q];
	printf("Enter elements in matrix:\n");
	for(i=0;i<p;i++){
		for(j=0;j<q;j++){
			printf("b[%d][%d]=",i,j);
			scanf("%d",&b[i][j]);
		}
	}

	//addition
//	int c[m][n];
//	for(i=0;i<m;i++){
```

```
//                  for(j=0;j<n;j++){
//                          c[i][j]=a[i][j]+b[i][j];
//                          printf("%d\t",c[i][j]);
//                  }
//                  printf("\n");
//          }
            //subtraction
//                  int c[m][n];
//          for(i=0;i<m;i++){
//                  for(j=0;j<n;j++){
//                          c[i][j]=a[i][j]-b[i][j];
//                          printf("%d\t",c[i][j]);
//                  }
//                  printf("\n");
//          }
//
            //Multiplication
//          int k;
//          int c[m][q];
//             for(i=0; i<m; i++) {
//       for(j=0; j<q; j++) {
//          c[i][j]=0;
//          for(k=0; k<n; k++) {
//             c[i][j] += a[i][k] * b[k][j];
//          }
//          printf("%d\t",c[i][j]);
//                  }
//       printf("\n");
//    }    }
```

```
#include<stdio.h>
#include<string.h>
int main(){
//	char s[50];
//	printf("Enter string:");
//	scanf("%s",s);
//	printf("Original String= %s\n",s);
//
	//length of string
//	int i=0,length=0;
//	while(s[i]!='\0'){
//		i++;
//		length++;
//	}
//	printf("Length of string:%d\n",length);
//
//	//reverse a string
//	i=0;
//	int j=length-1;
//	char temp;
//	while(i<j){
//		temp=s[i];
//		s[i]=s[j];
//		s[j]=temp;
//		i++;
//		j--;
//	}
//	printf("Reversed String= %s\n",s);
//
```

```
//      //Palindrome or not
//      int flag=0;
//      i=0,j=length-1;
//      while(i<j){
//              if(s[i]!=s[j]){
//                      flag=1;
//                      break;
//              }
//              i++;
//              j--;
//      }
//      if(flag==0){
//              printf("%s is plaindrome\n",s);
//      }
//      else{
//              printf("%s is noy palindrome\n",s);
//      }
//      //count letters,numbers,symbols
//      int letters=0;
//      int numbers=0;
//      int symbols=0;
//      i=0;
//      while(s[i]!='\0'){
//              if((s[i]>='A' && s[i]<='Z')||(s[i]>='a' && s[i]<='z')){
//                      letters++;
//              }
//              else if(s[i]>='0' && s[i]<='9'){
//                      numbers++;
//              }
```

```
//                  else{
//                          symbols++;
//                  }
//                  i++;
//
//          }
//          printf("Letters=%d\t numbers=%d\t symbols=%d\n",letters,numbers,symbols);
//
//          //count vowels and consonants
//          int vowels=0;
//          int consonants=0;
//          i=0;
//          while(s[i]!='\0'){
//                  if(s[i]=='a'|| s[i]=='e'|| s[i]=='i'|| s[i]=='o'||s[i]=='u'||s[i]=='A'|| s[i]=='E'||
s[i]=='I'|| s[i]=='O'||s[i]=='U'){
//                          vowels++;
//                  }
//                  else{
//                          consonants++;
//                  }
//                  i++;
//          }
//          printf("vowels=%d\t consonants=%d\n",vowels,consonants);
//
//          //uppercase to lowercase and vice versa
//          i=0;
//          while(s[i]!='\0'){
//                  if(s[i]>='A' && s[i]<='Z'){
//                          s[i]=s[i]+32;
//                  }
```

```
//                  else if(s[i]>='a' && s[i]<='z'){
//                          s[i]=s[i]-32;
//                  }
//                  i++;
//          }
//          printf("String = %s\n",s);


          //compare strings
//          char s1[50];
//          printf("Enter String:");
//          scanf("%s",s1);
//           i=0;
//          int j=0;
//          int flag=0;
//          while(s[i]!='\0' && s1[j]!='\0'){
//                  if(s[i]!=s1[j]){
//                          flag=1;
//                          break;
//                  }
//                  i++;
//                  j++;
//          }
//          if(flag==1){
//                  printf("Both strings are different\n");
//          }
//          else if(flag==0){
//                  printf("Both strings are same\n");
//          }
```

```
	//concatenate

//	while(s[i]!='\0'){
//		i++;
//	}
//	while(s1[j]!='\0'){
//		s[i]=s1[j];
//		i++;
//		j++;
//	}
//	s[i] = '\0';
//	printf("new string = %s\n",s);
//

//	//frequency of each character
//	int visit[length],count;
//	for(i=0;i<length;i++){
//		visit[i]=0;
//	}
//	i=0;
//	while(s[i]!='\0'){
//		if(visit[i]==1){
//			i++;
//			continue;
//		}
//		else{
//			count=1;
//			j=i+1;
//			while(s[j]!='\0'){
```

```
//                                  if(s[i]==s[j]){
//                                          count++;
//                                          visit[j]=1;
//                                  }
//                                  j++;
//
//                          }
//                                  printf("%c=%d\n",s[i],count);
//                  }
//                  i++;
//          }

          //remove duplicate character from string
//          int k=0,found=0;
//          char result[50];
//          while(s[i]!='\0'){
//                  found=0;
//                  j=i+1;
//                  while(s[j]!='\0'){
//                          if(s[i]==s[j]){
//                                  found=1;
//                                  break;
//                          }
//                          j++;
//                  }
//                  if(found==0){
//                          result[k]=s[i];
//                          k++;
//                  }
```

```
//                  i++;
//          }
//          printf("String = %s\n",result);

          //find first non repeating character
//          int found=0;
//                  while(s[i]!='\0'){
//                  found=0;
//                  j=0;
//                  while(s[j]!='\0'){
//                           if(i!=j && s[i]==s[j]){
//                                    found=1;
//                                    break;
//                           }
//                           j++;
//                  }
//                  if(found==0){
//                           printf("%c is first non repeating character\n",s[i]);
//                           break;
//                  }
//                  i++;
//          }

          //count words in string
//          char s[50];
//          printf("Enter string:");
//          gets(s);
//          printf("%s\n",s);
//          int words=0,i=0;
```

```
//        while(s[i]!='\0' ){
//                if(s[i]!=' ' && (s[i+1]==' '|| s[i+1]=='\0')){
//                        words++;
//                }
//                i++;
//        }
//        printf("Words=%d\n",words);



        //remove specific character
//        char ch;
//        int j,i=0;
//        printf("enter the element you want to remove:");
//        scanf("%c",&ch);
//        int k=0,found=0;
//        char result[50];
//        while(s[i]!='\0'){
//                found=0;
//                        if(s[i]!=ch){
//                                result[k]=s[i];
//                        k++;
//                        }
//                i++;
//        }
//        printf("String = %s\n",result);


        //string anagrams
//
//   char s1[50], s2[50];
```

```
//   printf("Enter first string: ");
//   scanf("%s", s1);
//   printf("Enter second string: ");
//   scanf("%s", s2);
//
//   int i = 0, j = 0, length1 = 0, length2 = 0;
//
//   while (s1[length1] != '\0')
//      length1++;
//   while (s2[length2] != '\0')
//      length2++;
//
//   if (length1 != length2) {
//      printf("Strings are not anagrams\n");
//      return 0;
//   }
//
//   int visit1[50] = {0}, visit2[50] = {0};
//   int count1[50] = {0}, count2[50] = {0};
//
//   // Counting characters in s1
//   for (i = 0; i < length1; i++) {
//      if (visit1[i] == 1)
//         continue;
//      int cnt = 1;
//      for (j = i + 1; j < length1; j++) {
//         if (s1[i] == s1[j]) {
//            cnt++;
//            visit1[j] = 1;
```

```
//         }
//      }
//      count1[i] = cnt;
//   }


   // Counting characters in s2
//   for (i = 0; i < length2; i++) {
//      if (visit2[i] == 1)
//         continue;
//      int cnt = 1;
//      for (j = i + 1; j < length2; j++) {
//         if (s2[i] == s2[j]) {
//            cnt++;
//            visit2[j] = 1;
//         }
//      }
//      count2[i] = cnt;
//   }
//
//   // Compare characters and counts
//   int matched = 1;
//   for (i = 0; i < length1; i++) {
//      int found = 0;
//      for (j = 0; j < length2; j++) {
//         if (s1[i] == s2[j] && count1[i] == count2[j]) {
//            found = 1;
//            break;
//         }
//      }
```

```
//      if (!found) {
//          matched = 0;
//          break;
//      }
//  }
//
//  if (matched)
//      printf("Strings are anagrams\n");
//  else
//      printf("Strings are not anagrams\n");
//
//  return 0;


          //ascending and descending order os string
  char str[50], temp;
  int i, j, len;
          printf("Enter a string:");
          scanf("%s",str);
   while (str[i] != '\0'){
     len++;
     i++;
  }



  for (i = 0; i < len - 1; i++) {
     for (j = i + 1; j < len; j++) {
        if (str[i] > str[j]) {
           temp = str[i];
           str[i] = str[j];
```

```
            str[j] = temp;
        }
      }
    }

    printf("String in ascending order: %s\n", str);


    for (i = 0; i < len - 1; i++) {
      for (j = i + 1; j < len; j++) {
        if (str[i] < str[j]) {
          temp = str[i];
          str[i] = str[j];
          str[j] = temp;
        }
      }
    }

    printf("String in descending order: %s\n", str);

}
```

```
//Key ASCII Ranges for Strings
//
```

```
//Uppercase Letters
//
//A ? 65
//
//Z ? 90
//
//Lowercase Letters
//
//a ? 97
//
//z ? 122
//
//Digits
//
//0 ? 48
//
//9 ? 57
//
//Space and Special Characters
//
//Space ? 32
//
//! ? 33
//
//. ? 46
//
//, ? 44
//
//? ? 63
```

```c
#include<stdio.h>
int main(){
	int n,i,fact;
	printf("Enter a number:");
	scanf("%d",&n);
	for(i=n;i>=1;i--){
		fact=fact*i;
	}
	printf("Factorial of %d = %d\n",n,fact);
}

//#include<stdio.h>
//
//// Recursive function for factorial
//int factorial(int n){
//   if(n == 0 || n == 1)   // base condition
//      return 1;
//   else
//      return n * factorial(n - 1);  // recursive step
//}
//
//int main(){
//   int n;
//   printf("Enter a number: ");
//   scanf("%d", &n);
//
//   printf("Factorial of %d = %d\n", n, factorial(n));
//   return 0;
//}
```

```
#include<stdio.h>
int main(){
	int i,n,first=1,second=1,temp;
	printf("Enter range of fibonacci series:");
	scanf("%d",&n);
	printf("%d\t%d\t",first,second);
	for(i=1;i<=n-2;i++){
		temp=first+second;
		first=second;
		second=temp;
		printf("%d\t",temp);
	}
}

//#include<stdio.h>
//
//// recursive function for fibonacci
//int fibonacci(int n){
//   if(n == 0)   // base case
//      return 0;
//   else if(n == 1)
//      return 1;
//   else
//      return fibonacci(n-1) + fibonacci(n-2);  // recursive calls
//}
//
//int main(){
//   int n, i;
//   printf("Enter number of terms: ");
```

```
//    scanf("%d", &n);
//
//    printf("Fibonacci series up to %d terms:\n", n);
//    for(i = 0; i < n; i++){
//        printf("%d\t", fibonacci(i));
//    }
//    return 0;
//}

#include<stdio.h>
int main(){
        int a,b,temp;
        printf("Enter two numbers:");
        scanf("%d %d",&a,&b);
        while(a!=0){
                temp=a;
                a=a%b;
                b=temp;
        }
        printf("GCD is %d\n",b);
}


#include<stdio.h>
int main(){
        int a,b,temp;
        printf("Enter two numbers:");
        scanf("%d %d",&a,&b);
        if(a>b){
```

```
		temp=a;
	}
	else{
		temp=b;
	}
	while(a!=b){
		if(temp%a==0 && temp%b==0){
			break;
		}
		temp++;
	}

	printf("LCM is %d\n",temp);
}



#include<stdio.h>
int main(){
	int i,j,n,flag=0;
	printf("Enter no. of numbers:(from 2 to n)");
	scanf("%d",&n);
	printf("List of prime numbers till %d\n",n);
	for(i=2;i<=n;i++){
		flag=0;
		for(j=2;j<i-1;j++){
		if(i%j==0){
			flag=1;
			break;
		}
```

```
        }
                if(flag==0){
                printf("%d\t",i);
}
}
}

#include<stdio.h>
int main(){
        int i,n,flag=0;
        printf("Enter a number:");
        scanf("%d",&n);
        for(i=2;i<n-1;i++){
                if(n%i==0){
                        flag=1;
                        break;
                }
        }
        if(flag==0){
                printf("number is prime");
        }
        else{
                printf("Number is non prime");
        }
}
```

```
#include<stdio.h>
#include<math.h>
//int fact(int n){
//   int i, f = 1;
//   for(i = 1; i <= n; i++)
//      f *= i;
//   return f;
//}
int main(){
        int temp, n,rev=0,el;
        printf("Enter a number:");
        scanf("%d",&n);

        //reverse no.
        temp=n;
//      while(n!=0){
//              el=n%10;
//              rev=rev*10+el;
//              n=n/10;
//      }
//      printf("Reversed no. is %d\n",rev);

        //palindrome of no.
//      if(temp==rev){
//              printf("No. is plaindrome\n");
//      }
//      else{
//              printf("No. is not a plaindrome\n");
//      }
```

```
        //armstrong no.
Sum of each digit raised to the power of (number of digits)=number itself
//      int sum,count=n,digit=0;
//      while(count!=0){
//              digit++;
//              count=count/10;
//      }
//      while(n!=0){
//              el=n%10;
//              sum=sum+pow(el,digit);
//              n=n/10;
//      }
//      if(sum==temp){
//              printf("No. is armstrong\n");
//      }
//      else{
//              printf("No. is not armstrong\n");
//      }

        //Strong no.  A number is strong if sum of factorials of digits = number.

//      int sum=0;
//  while(n != 0){
//      el = n % 10;
//      sum += fact(el);
//      n = n / 10;
//  }
//  if(sum == temp)
//      printf("Strong number\n");
```

```
//    else
//       printf("Not strong\n");


            //sum of digit
            int sum=0;
            while(n!=0){
                        el=n%10;
                        sum=sum+el;
                        n=n/10;
              }
              printf("Sum of no. is:%d\n",sum);
}


#include<stdio.h>
int main(){
            int n,i,j,flag,temp,count=0;
            printf("enter the size of array:");
            scanf("%d",&n);
            int a[n];
            for(i=0;i<n;i++){
                        printf("a[%d]=",i);
                        scanf("%d",&a[i]);
            }
            printf("Array before sorting:\t");
            for(i=0;i<n;i++){
                        printf("%d\t",a[i]);
            }
            for(i=0;i<n-1;i++){
                        flag=0;
```

```
        for(j=0;j<n-1-i;j++){
                if(a[j]>a[j+1]){
                        temp=a[j];
                        a[j]=a[j+1];
                        a[j+1]=temp;
                        flag=1;
                        count++;
                }
        }
        if(flag==0){
                break;
        }
}
printf("\nArray after sorting:(ascending)\t");
for(i=0;i<n;i++){
        printf("%d\t",a[i]);
}
        for(i=0;i<n-1;i++){
        flag=0;
        for(j=0;j<n-1-i;j++){
                if(a[j]<a[j+1]){
                        temp=a[j];
                        a[j]=a[j+1];
                        a[j+1]=temp;
                        flag=1;
                        count++;
                }
        }
        if(flag==0){
```

```
                        break;
                }
        }
        printf("\nArray after sorting:(descending)\t");
        for(i=0;i<n;i++){
                printf("%d\t",a[i]);
        }
//      printf("\nNo. of swaps in sorting:%d\n",count);
}

//#include<stdio.h>
//#include<string.h>
//int main(){
//      int n,i,j,flag,temp,count=0;
//      char s[50];
//      printf("Enter string:");
//      scanf("%s",s);
//      n=strlen(s);
//      printf("String before sorting:\t");
//      printf("%s",s);
//      for(i=0;i<n-1;i++){
//              flag=0;
//              for(j=0;j<n-1-i;j++){
//                      if(s[j]>s[j+1]){
//                              temp=s[j];
//                              s[j]=s[j+1];
//                              s[j+1]=temp;
//                              flag=1;
//                              count++;
```

```
//                              }
//                  }
//                  if(flag==0){
//                              break;
//                  }
//          }
//          printf("\nString after sorting:\t");
//          printf("%s",s);
//          printf("\nNo. of swaps in sorting:%d\n",count);
//}

//#include<stdio.h>
//int main(){
//          float key;
//          int n,i,j,count=0;;
//          printf("enter the size of array:");
//          scanf("%d",&n);
//          float a[n];
//          for(i=0;i<n;i++){
//                  printf("a[%d]=",i);
//                  scanf("%f",&a[i]);
//          }
//          printf("Array before sorting:\t");
//          for(i=0;i<n;i++){
//                  printf("%.2f\t",a[i]);
//          }
//          for(i=1;i<n;i++){
//                  key=a[i];
//                  j=i-1;
```

```
//                  while(j>=0 && a[j]>key){
//                          a[j+1]=a[j];
//                          j--;
//                          count++;
//                  }
//                  a[j+1]=key;
//          }
//          printf("\nArray after sorting:(ascending)");
//          for(i=0;i<n;i++){
//                  printf("%.2f\t",a[i]);
//          }
//          printf("\nNo. of shifts:%d\n",count);
//          for(i=1;i<n;i++){
//                  key=a[i];
//                  j=i-1;
//                  while(j>=0 && a[j]<key){
//                          a[j+1]=a[j];
//                          j--;
//                          count++;
//                  }
//                  a[j+1]=key;
//          }
//          printf("\nArray after sorting:(descending)");
//          for(i=0;i<n;i++){
//                  printf("%.2f\t",a[i]);
//          }
//}
```

```
#include<stdio.h>
#include<string.h>
int main(){
	int n,i,j,flag,temp,count=0;
	char s[50],key;
	printf("Enter string:");
	scanf("%s",s);
	n=strlen(s);
	printf("String before sorting:\t");
	printf("%s",s);
		for(i=1;i<n;i++){
		key=s[i];
		j=i-1;
		while(j>=0 && s[j]>key){
			s[j+1]=s[j];
			j--;
			count++;
		}
		s[j+1]=key;
	}
	printf("\nString after sorting:\t");
	printf("%s",s);
	printf("\nNo. of swaps in sorting:%d\n",count);
}
```

```
//#include<stdio.h>
//void print(int a[],int n){
//	int i;
//	for(i=0;i<n;i++){
//		printf("%d\t",a[i]);
//	}
//}
//void quicksort(int a[],int lb,int ub){
//	int pos;
//	if(lb<ub){
//		pos=partition(a,lb,ub);
//		quicksort(a,lb,pos-1);
//		quicksort(a,pos+1,ub);
//	}
//}
//int partition(int a[], int lb,int ub){
//	int pivot=a[lb];
//	int start=lb;
//	int end=ub;
//	int temp;
//	while(start<end){
//		while(a[start]<=pivot){
//			start++;
//		}
//		while(a[end]>pivot){
//			end--;
//		}
//		if(start<end){
//			temp=a[start];
```

```
//                          a[start]=a[end];
//                          a[end]=temp;
//                  }
//          }
//          temp=a[lb];
//          a[lb]=a[end];
//          a[end]=temp;
//          return end;
//}
//int main(){
//          int n;
//          printf("Enter the size of array:");
//          scanf("%d",&n);
//          int a[n],i;
//          printf("Enter array elements:");
//          for(i=0;i<n;i++){
//                  printf("a[%d]=",i);
//                  scanf("%d",&a[i]);
//          }
//          int lb=0;
//          int ub=n-1;
//          printf("Array before sorting:");
//          print(a,n);
//          quicksort(a,lb,ub);
//          printf("\nArray after sorting");
//          print(a,n);
//}
//
```

```
//#include<stdio.h>
//void print(int a[],int n){
//	int i;
//	for(i=0;i<n;i++){
//		printf("%d\t",a[i]);
//	}
//}
//void quicksort(int a[],int lb,int ub){
//	int pos;
//	if(lb<ub){
//		pos=partition(a,lb,ub);
//		quicksort(a,lb,pos-1);
//		quicksort(a,pos+1,ub);
//	}
//}
//int partition(int a[], int lb,int ub){
//	int pivot=a[lb];
//	int start=lb;
//	int end=ub;
//	int temp;
//	while(start<end){
//		while(a[start]>=pivot){
//			start++;
//		}
//		while(a[end]<pivot){
//			end--;
//		}
//		if(start<end){
//			temp=a[start];
```

```
//                          a[start]=a[end];
//                          a[end]=temp;
//                  }
//          }
//          temp=a[lb];
//          a[lb]=a[end];
//          a[end]=temp;
//          return end;
//}
//int main(){
//          int n;
//          printf("Enter the size of array:");
//          scanf("%d",&n);
//          int a[n],i;
//          printf("Enter array elements:");
//          for(i=0;i<n;i++){
//                  printf("a[%d]=",i);
//                  scanf("%d",&a[i]);
//          }
//          int lb=0;
//          int ub=n-1;
//          printf("Array before sorting:");
//          print(a,n);
//          quicksort(a,lb,ub);
//          printf("\nArray after sorting");
//          print(a,n);
//}
```

```
//#include<stdio.h>
//#include<string.h>
//void quicksort(char *s,int lb,int ub){
//	int pos;
//	if(lb<ub){
//		pos=partition(s,lb,ub);
//		quicksort(s,lb,pos-1);
//		quicksort(s,pos+1,ub);
//	}
//}
//int partition(char* s, int lb,int ub){
//	int pivot=s[lb];
//	int start=lb;
//	int end=ub;
//	int temp;
//	while(start<end){
//		while(s[start]<=pivot){
//			start++;
//		}
//		while(s[end]>pivot){
//			end--;
//		}
//		if(start<end){
//			temp=s[start];
//			s[start]=s[end];
//			s[end]=temp;
//		}
//	}
//	temp=s[lb];
```

```
//        s[lb]=s[end];
//        s[end]=temp;
//        return end;
//}
//int main(){
//        char s[50],i;
//        printf("Enter string:");
//        scanf("%s",s);
//        int n=strlen(s);
//        int lb=0;
//        int ub=n-1;
//        printf("string before sorting:");
//        printf("%s",s);
//        quicksort(s,lb,ub);
//        printf("string before sorting:");
//        printf("%s",s);
//}



#include<stdio.h>

int quicksort(int a[],int lb,int ub,int k){
        if(lb<=ub){

        int pos;
        pos=partition(a,lb,ub);

        if(pos==k-1){
                return a[pos];
```

```
        }
        else if(pos>k-1){

                return quicksort(a,lb,pos-1,k);
        }
        else if(pos<k-1){
                return quicksort(a,pos+1,ub,k);
        }
}
}

int partition(int a[], int lb,int ub){
        int pivot=a[lb];
        int start=lb;
        int end=ub;
        int temp;
        while(start<end){
                while(a[start]<=pivot){
                        start++;
                }
                while(a[end]>pivot){
                        end--;
                }
                if(start<end){
                        temp=a[start];
                        a[start]=a[end];
                        a[end]=temp;
                }
        }
```

```
        temp=a[lb];
        a[lb]=a[end];
        a[end]=temp;
        return end;
}
int main(){
        int n;
        printf("Enter the size of array:");
        scanf("%d",&n);
        int a[n],i;
        printf("Enter array elements:");
        for(i=0;i<n;i++){
                printf("a[%d]=",i);
                scanf("%d",&a[i]);
        }
        int lb=0;
        int ub=n-1;
        int k,result;
        printf("Enter the kth-smallest element:");
        scanf("%d",&k);
        result=quicksort(a,lb,ub,k);
        printf("%d smallest element is: %d\n",k,result);
}
```

```
//#include<stdio.h>
//int main(){
//	float temp;
//	int n,i,j;
//	printf("enter the size of array:");
//	scanf("%d",&n);
//	float a[n];
//	for(i=0;i<n;i++){
//		printf("a[%d]=",i);
//		scanf("%f",&a[i]);
//	}
//	printf("Array before sorting:\t");
//	for(i=0;i<n;i++){
//		printf("%.2f\t",a[i]);
//	}
//	for(i=0;i<n-1;i++){
//		for(j=i+1;j<n;j++){
//			if(a[i]>a[j]){
//				temp=a[i];
//				a[i]=a[j];
//				a[j]=temp;
//			}
//		}
//	}
//	printf("\nArray after sorting:(ascending)");
//	for(i=0;i<n;i++){
//		printf("%.2f\t",a[i]);
//	}
//	for(i=0;i<n-1;i++){
```

```
//                  for(j=i+1;j<n;j++){
//                          if(a[i]<a[j]){
//                                  temp=a[i];
//                                  a[i]=a[j];
//                                  a[j]=temp;
//                          }
//                  }
//          }
//          printf("\nArray after sorting:(descending)");
//          for(i=0;i<n;i++){
//                  printf("%.2f\t",a[i]);
//          }
//}

//#include<stdio.h>
//#include<string.h>
//int main(){
//          int n,i,j,flag,count=0;
//          char s[50],temp;
//          printf("Enter string:");
//          scanf("%s",s);
//          n=strlen(s);
//          printf("String before sorting:\t");
//          printf("%s",s);
//          for(i=0;i<n-1;i++){
//                  for(j=i+1;j<n;j++){
//                                  if(s[i]>s[j]){
//                                  temp=s[i];
//                                  s[i]=s[j];
```

```
//                                      s[j]=temp;
//                              }
//                      }
//              }
//      printf("\nString after sorting:\t");
//      printf("%s",s);
//
//}

#include<stdio.h>
int main(){
        float temp;
        int n,i,j,k;
        printf("enter the size of array:");
        scanf("%d",&n);
        float a[n];
        for(i=0;i<n;i++){
                printf("a[%d]=",i);
                scanf("%f",&a[i]);
        }
        printf("Array before sorting:\t");
        for(i=0;i<n;i++){
                printf("%.2f\t",a[i]);
        }
        printf("Enter the k-th smallest element you want:");
        scanf("%d",&k);
        for(i=0;i<k;i++){
                for(j=i+1;j<n;j++){
                        if(a[i]>a[j]){
```

```
                                temp=a[i];
                                a[i]=a[j];
                                a[j]=temp;
                        }
                }
        }
        printf("\nArray after sorting:(ascending)");
        for(i=0;i<n;i++){
                printf("%.2f\t",a[i]);
        }
        printf("\n%d smallest element is : %.2f\n",k,a[k-1]);
}

//NAME: DHARA DHUVAVIYA USN NO.:CS23181
#include<stdio.h>
void print_array(int arr[],int n);
void mergesort(int arr[],int l,int r);
void merge(int arr[],int l,int m,int r);

int main(){
        int arr[50],n,i;
        printf("enter the size of array:");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf("arr[%d]=",i);
                scanf("%d",&arr[i]);
        }
        printf("Array before sorting:\n");
        print_array(arr,n);
```

```
	mergesort(arr,0,n-1);
	printf("Array after sorting:\n");
	print_array(arr,n);
	return 0;
}

void print_array(int arr[],int n){
	int i;
	for(i=0;i<n;i++){
		printf("%d\t",arr[i]);
		printf("\n");
	}
}

void mergesort(int arr[],int l,int r){
	int m;
	if(l<r){
		m=l+(r-l)/2;    //find the middle index
		mergesort(arr,l,m); //recursive calling   //spliting left index element
		mergesort(arr,m+1,r);                                                    //spliting
right index element
		merge(arr,l,m,r);
	}
}

void merge(int arr[],int l,int m,int r){
	int i,j,k,n1,n2;
	n1=m-l+1;   //doubt
	n2=r-m;
```

```
int L[n1],R[n2];
//copy data into temporary arrays
for(i=0;i<n1;i++){
        L[i]=arr[l+i];   //doubt
}
for(j=0;j<n2;j++){
        R[j]=arr[m+1+j];        //doubt

}
i=0;
j=0;
k=0;
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++;
            }
}


#include <stdio.h>
#include <string.h>

typedef struct {
   char name[50];
   int marks;
} Student;

void merge(Student arr[], int l, int m, int r) {
   int n1 = m - l + 1;
   int n2 = r - m;

   Student L[n1], R[n2];

   for (int i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (int j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];

   int i = 0, j = 0, k = l;
   while (i < n1 && j < n2) {
      if (L[i].marks <= R[j].marks)
```

```
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }

    while (i < n1)
        arr[k++] = L[i++];
    while (j < n2)
        arr[k++] = R[j++];
}

void mergeSort(Student arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

int main() {
    Student s[] = {{"Dhara", 90}, {"Ayush", 75}, {"Riya", 88}, {"Karan", 92}};
    int n = 4;

    mergeSort(s, 0, n - 1);

    printf("Students sorted by marks:\n");
    for (int i = 0; i < n; i++)
        printf("%s - %d\n", s[i].name, s[i].marks);
```

```
    return 0;
}



#include <stdio.h>

void mergeSortedArrays(int A[], int n1, int B[], int n2, int C[]) {
    int i = 0, j = 0, k = 0;

    while (i < n1 && j < n2) {
        if (A[i] <= B[j])
            C[k++] = A[i++];
        else
            C[k++] = B[j++];
    }

    while (i < n1)
        C[k++] = A[i++];
    while (j < n2)
        C[k++] = B[j++];
}

int main() {
    int A[] = {1, 3, 5, 7};
    int B[] = {2, 4, 6, 8};
    int n1 = 4, n2 = 4;
    int C[n1 + n2];
```

```
    mergeSortedArrays(A, n1, B, n2, C);

    printf("Merged Array: ");
    for (int i = 0; i < n1 + n2; i++)
        printf("%d ", C[i]);
    printf("\n");
    return 0;
}
```

```
#include<stdio.h>
        #define n 5
        int stack[n];
        int top=-1;
void push(){
        if(top==n-1){
                printf("Stack overflow\n");
        }
        else{
                        top++;
                int el;
                printf("Enter the element:");
                scanf("%d",&el);
                stack[top]=el;
        }
}
void pop(){
        if(top==-1){
                printf("Stack is empty\n");
```

```
        }
        else{
                printf("%d element is poped out\n",stack[top]);
                top--;
        }
}
void peek(){
        if(top==-1){
                printf("Stack is empty\n");
        }
        else{
                printf("%d is topmost element of stack\n",stack[top]);
        }
}
void display(){
        if(top==-1){
                printf("Stack is empty\n");
        }
        else{
                int i;
                printf("Stack elements are:");
                for(i=top;i>=0;i--){
                        printf("%d\t",stack[i]);
                }
                printf("\n");
        }
}
int main(){
        int ch;
```

```
        while(1){

        printf("Operations\n1.Push()\n2.Pop()\n3.Peek()\n4.Display()\n5.Exit()\n");
                printf("Enter the options:");
                scanf("%d",&ch);
                switch(ch){
                        case 1:
                                push();
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                peek();
                                break;
                        case 4:
                                display();
                                break;
                        case 5:
                                printf("Exiting....\n");
                                return 1;
                        default:
                                printf("Operation is not valid\n");
                                break;
                }
        }
}
```

```
#include<stdio.h>
#include<ctype.h>
# define size 20
int stack[size];
int top=-1;
void push(char c){
	top++;
	stack[top]=c;
}
char pop(){
	return stack[top--];
}
int precedence(char c){
	if(c=='^') return 3;
	if(c=='/' || c=='*') return 2;
	if(c=='+'|| c=='-') return 1;
	else return 0;
}
char infixtopostfix(char *infix){
	int i=0,k=0;
	char result[size];
	char c;
	while(infix[i]!='\0'){
		c=infix[i];
		if(isalnum(infix[i])){
			result[k]=c;
			k++;
		}
		else if(c=='('){
```

```
                push(c);
            }
            else if(c==')'){
                while(top!=-1 && stack[top]!='('){
                    result[k]=pop();
                    k++;
                }
                pop();
            }
            else{
                while(top!=-1 && precedence(stack[top])>=precedence(c)){
                    result[k]=pop();
                    k++;
                }
                push(c);
            }
            i++;
        }
        while(top!=-1){
            result[k]=pop();
            k++;
        }
        printf("%s is postfix expression\n",result);
}
int main(){
        char infix[size];
        printf("Enter the infix expression:");
        scanf("%s",infix);
        infixtopostfix(infix);}
```

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
# define size 20
int stack[size];
int top=-1;
void push(char c){
        top++;
        stack[top]=c;
}
char pop(){
        return stack[top--];
}
int precedence(char c){
        if(c=='^') return 3;
        if(c=='/' || c=='*') return 2;
        if(c=='+'|| c=='-') return 1;
        else return 0;
}
void reverse(char *exp){
        int len=strlen(exp);
        int i=0,j=len-1,temp;
        while(i<j){
                temp=exp[i];
                exp[i]=exp[j];
                exp[j]=temp;
                i++;
                j--;
        }
```

```
        for(i=0;i<len;i++){
                if(exp[i]=='('){
                        exp[i]=')';
                }
                else if(exp[i]==')'){
                        exp[i]='(';
                }
        }
        }

char infixtoprefix(char *infix){
        reverse(infix);
        int i=0,k=0;
        char result[size];
        char c;
        while(infix[i]!='\0'){
                c=infix[i];
                if(isalnum(infix[i])){
                        result[k]=c;
                        k++;
                }
                else if(c=='('){
                        push(c);
                }
                else if(c==')'){
                        while(top!=-1 && stack[top]!='('){
                                result[k]=pop();
                                k++;
                        }
```

```
                pop();
            }
            else{
                while(top!=-1 && precedence(stack[top])>=precedence(c)){
                    result[k]=pop();
                    k++;
                }
                push(c);
            }
            i++;
        }
        while(top!=-1){
            result[k]=pop();
            k++;
        }
        reverse(result);
        printf("%s is prefix expression\n",result);
}
int main(){
        char infix[size];
        printf("Enter the infix expression:");
        scanf("%s",infix);
        infixtoprefix(infix);
}
```

```
#include<stdio.h>
#include<ctype.h>
#include<math.h>
# define size 20
int stack[size];
int top=-1;
void push(char c){
        top++;
        stack[top]=c;
}
char pop(){
        return stack[top--];
}
float evaluate(char *postfix){
        int i=0,op1,op2;
        while(postfix[i]!='\0'){
                if(isdigit(postfix[i])){
                        push(postfix[i]-'0');
        }
        else{
                op2=pop();
                op1=pop();
                switch(postfix[i]){
                        case '+':
                                push(op1+op2);
                                break;
                        case '-':
                                push(op1-op2);
                                break;
```

```
                        case '*':
                                push(op1*op2);
                                break;
                        case '/':
                                push(op1/op2);
                                break;
                        case '^':
                                push(pow(op1,op2));
                                break;
                }
        }
        i++;
}
return pop();
}
int main(){
        char postfix[size];
        printf("Enter the postfix expression:");
        gets(postfix);
        printf("Result=%.2f\n",evaluate(postfix));
}

#include<stdio.h>
#include<ctype.h>
#include<math.h>
#include<string.h>
# define size 20
float stack[size];
int top=-1;
```

```
void push(float c){
        top++;
        stack[top]=c;
}
float pop(){
        return stack[top--];
}
float evaluate(char *prefix){
        int i=0;
        float op1,op2;
        for(i=strlen(prefix)-1;i>=0;i--){
            if (prefix[i] == ' ') continue;
            if(isdigit(prefix[i])){
                push(prefix[i]-'0');
        }
        else{
            op2=pop();
            op1=pop();
            switch(prefix[i]){
                case '+':
                    push(op1+op2);
                    break;
                case '-':
                    push(op1-op2);
                    break;
                case '*':
                    push(op1*op2);
                    break;
                case '/':
```

```
                push(op1/op2);
                break;
            case '^':
                push(pow(op1,op2));
                break;
        }
    }
}
return pop();
}
int main(){
    char prefix[size];
    printf("Enter the prefix expression:");
    scanf("%s",prefix);
    printf("Result=%.2f\n",evaluate(prefix));
}

#include<stdio.h>
    #define n 20
    char stack[n];
    int top=-1;
void push(char c){
    if(top==n-1){
        printf("Stack overflow\n");
    }
    else{
        top++;
        stack[top]=c;
    }
```

```
}
void pop(){
	if(top==-1){
		printf("Stack is empty\n");
	}
	else{
		printf("%c",stack[top]);
		top--;
	}
}
void reverse(char *s)
{
	int i=0;
	while(s[i]!='\0'){
		push(s[i]);
		i++;
	}
	i=0;
	while(s[i]!='\0'){
		pop();
		i++;
	}
}
int main(){
	char s[n];
	printf("Enter string:");
	scanf("%s",s);
	reverse(s);
	}
```

```
#include <stdio.h>
#define n 20
char stack[n];
int top = -1;
void push(char c) {
   if (top == n - 1) {
      printf("Stack overflow\n");
   } else {
      top++;
      stack[top] = c;
   }
}
char pop() {
   if (top == -1) {
      printf("Stack is empty!\n");
      return '\0';
   } else {
      return stack[top--];
   }
}
int match(char open, char close) {
   return (open == '(' && close == ')') ||
         (open == '{' && close == '}') ||
         (open == '[' && close == ']');
}
int evaluate(char *s) {
   char c;
   int i = 0;
```

```
    while (s[i] != '\0') {
        c = s[i];
        if (c == '(' || c == '{' || c == '[') {
            push(c);
        } else if (c == ')' || c == '}' || c == ']') {
            if (top == -1 || !match(pop(), c)) {
                return 0;
            }
        }
        i++;
    }
    return top == -1;
}
int main() {
    char s[n];
    printf("Enter the expression: ");
    scanf("%s", s);
    if (evaluate(s)) {
        printf("It's balanced!\n");
    } else {
        printf("It's not balanced!\n");
    }
   return 0;
}
```

```
#include<stdio.h>
#define n 10
int queue[n];
int front=-1,rear=-1;
void enqueue(){
	if(rear==n-1){
		printf("Queue is full!\n");
		return;
	}
	else{
		int el;
		printf("Enter the element:");
		scanf("%d",&el);
		if(front==-1){
			front=0;
		}
		rear++;
		queue[rear]=el;
	}
}
void dequeue(){
	if(front==-1 || front>rear){
		printf("Queue is empty!\n");
		return;
	}
	else{
		printf("%d is dequeued from queue\n",queue[front]);
		front++;
	}
```

```
}
void peek(){
	if(front==-1 || front>rear){
		printf("Queue is empty!\n");
		return;
	}
	else{
		printf("%d is peek element of queue\n",queue[front]);
	}
}
void display(){
	if(front==-1 || front>rear){
		printf("Queue is empty!\n");
		return;
	}
	else{
		int i;
		printf("elements in queue are:");
		for(i=front;i<=rear;i++){
			printf("%d\t",queue[i]);
		}
	}
}
int main(){
	int ch;
	while(1){

	printf("Operations\n1.Inqueue()\n2.Dequeue()\n3.Peek()\n4.Display()\n5.Exit()\n
Enter the option:");
		scanf("%d",&ch);
```

```
            switch(ch){
                case 1:
                    enqueue();
                    break;
                case 2:
                    dequeue();
                    break;
                case 3:
                    peek();
                    break;
                case 4:
                    display();
                    break;
                case 5:
                    printf("Exitingg...\n");
                    return;
                default:
                    printf("The option is invalid\n");
                    break;
            }
        }
}
```

```
#include<stdio.h>
#include <stdio.h>
#define SIZE 50
int queue[SIZE];
int front = -1, rear = -1;
void enqueue(int num) {
   if (rear == SIZE - 1) {
      printf("Queue is full\n");
      return;
   }
   if (front == -1) front = 0;
   queue[++rear] = num;
}
int dequeue() {
   if (front == -1 || front > rear) {
      return -1;
   }
   return queue[front++];
}
void binary(int num){
enqueue(1);
int i;
for(i=1;i<=num;i++){
            int n=dequeue();
            printf("%d\t",n);
            enqueue(n*10);
            enqueue(n*10+1);
}
}
```

```
int main(){
        int num;
        printf("How many binary numbers should be generated:");
        scanf("%d",&num);
        binary(num);
}



#include<stdio.h>
#include <string.h>
#define SIZE 50
char queue[SIZE][20];
int front = -1, rear = -1;
void enqueue(char *c) {
   if (rear == SIZE - 1) {
      printf("Queue is full\n");
      return;
   }
   if (front == -1) front = 0;
   strcpy(queue[++rear],c);
}
char* dequeue() {
   if (front == -1 || front > rear) {
      return -1;
   }
   return queue[front++];
}
void binary(int num){
char temp[20];
```

```
enqueue("1");
int i;
for(i=1;i<=num;i++){
        char *n=dequeue();
        printf("%s\t",n);
        strcpy(temp, n);
        strcat(temp,"0");
        enqueue(temp);
        strcpy(temp, n);
        strcat(temp,"1");
        enqueue(temp);
}
}
int main(){
        int num;
        printf("How many binary numbers should be generated:");
        scanf("%d",&num);
        binary(num);
}
```

```
#include<stdio.h>
#define n 10
int queue[n];
int front=-1,rear=-1;
void enqueue(){
        if((rear+1)%n==front){
                printf("Queue is full\n");
                return;
```

```
        }
        else{
                int el;
                printf("Enter element:");
                scanf("%d",&el);
                if(front==-1){
                        front=0;
                }
                rear=(rear+1)%n;
                queue[rear]=el;
        }
}
void frontdequeue(){
        if(front==-1){
                printf("Queue is empty\n");
        }
        printf("%d is dequeued from queue\n",queue[front]);
        if(front==rear){
                front=rear=-1;
        }
        else{
                front=(front+1)%n;
                }
}
void reardequeue(){
        if(front==-1){
                printf("Queue is empty\n");
        }
        printf("%d is dequeued from queue\n",queue[rear]);
```

```
        if(front==rear){
                front=rear=-1;
        }
        else{
                rear=(rear-1+n)%n;
                }
}
void display(){
        if(front==-1){
                printf("Queue is empty\n");
        }
        else{
                int i=front;
                printf("Queue elements are:");
                while(i!=rear){
                        printf("%d\t",queue[i]);
                        i=(i+1)%n;
                }
                 printf("%d\n", queue[rear]);
        }
}
int main(){
        int ch;
        while(1){

        printf("Operations\n1.Inqueue()\n2.Front_Dequeue()\n3.Rear_Dequeue()\n4.Dis
play()\n5.Exit()\nEnter the option:");
                scanf("%d",&ch);
                switch(ch){
                        case 1:
```

```
                enqueue();
                break;
            case 2:
                frontdequeue();
                break;
            case 3:
                reardequeue();
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exitingg...\n");
                return;
            default:
                printf("The option is invalid\n");
                break;
        }
    }
}


#include<stdio.h>
#define n 10
int queue[n];
int front=-1,rear=-1;
void rearenqueue(){
    if((rear+1)%n==front){
        printf("Queue is full\n");
```

```
                return;
        }
        else{
                int el;
                printf("Enter element:");
                scanf("%d",&el);
                if(front==-1){
                        front=0;
                }
                rear=(rear+1)%n;
                queue[rear]=el;
        }
}
void dequeue(){
        if(front==-1){
                printf("Queue is empty\n");
        }
        printf("%d is dequeued from queue\n",queue[front]);
        if(front==rear){
                front=rear=-1;
        }
        else{
                front=(front+1)%n;
                }
}
void frontenqueue(){
                if((rear+1)%n==front){
                printf("Queue is full\n");
                return;
```

```
        }
        else{
                int el;
                printf("Enter element:");
                scanf("%d",&el);
                if(front==-1){
                        front=0;
                }
                front = (front - 1 + n) % n;
                queue[front]=el;
        }
}
void display(){
        if(front==-1){
                printf("Queue is empty\n");
        }
        else{
                int i=front;
                printf("Queue elements are:");
                while(i!=rear){
                        printf("%d\t",queue[i]);
                        i=(i+1)%n;
                }
                 printf("%d\n", queue[rear]);
        }
}
int main(){
        int ch;
        while(1){
```

```
        printf("Operations\n1.Rear_enqueue()\n2.Front_enqueue()\n3.Dequeue()\n4.Dis
play()\n5.Exit()\nEnter the option:");
                scanf("%d",&ch);
                switch(ch){
                        case 1:
                                rearenqueue();
                                break;
                        case 2:
                                frontenqueue();
                                break;
                        case 3:
                                dequeue();
                                break;
                        case 4:
                                display();
                                break;
                        case 5:
                                printf("Exitingg...\n");
                                return;
                        default:
                                printf("The option is invalid\n");
                                break;
                }
        }
}
```

```
#include<stdio.h>
#define n 10
int queue[n];
int front=-1,rear=-1;
void enqueue(){
	if((rear+1)%n==front){
		printf("Queue is full\n");
		return;
	}
	else{
		int el;
		printf("Enter element:");
		scanf("%d",&el);
		if(front==-1){
			front=0;
		}
		rear=(rear+1)%n;
		queue[rear]=el;
	}
}
void dequeue(){
	if(front==-1){
		printf("Queue is empty\n");
	}
	printf("%d is dequeued from queue\n",queue[front]);
	if(front==rear){
		front=rear=-1;
	}
	else{
```

```
                front=(front+1)%n;
                }
}
void peek(){
        if(front==-1){
                printf("Queue is empty\n");
        }
        else{
                printf("%d is topmost element\n",queue[front]);
        }
}
void display(){
        if(front==-1){
                printf("Queue is empty\n");
        }
        else{
                int i=front;
                printf("Queue elements are:");
                while(i!=rear){
                        printf("%d\t",queue[i]);
                        i=(i+1)%n;
                }
        }
}
int main(){
        int ch;
        while(1){

        printf("Operations\n1.Inqueue()\n2.Dequeue()\n3.Peek()\n4.Display()\n5.Exit()\n
Enter the option:");
```

```
                scanf("%d",&ch);
                switch(ch){
                        case 1:
                                enqueue();
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                peek();
                                break;
                        case 4:
                                display();
                                break;
                        case 5:
                                printf("Exitingg...\n");
                                return;
                        default:
                                printf("The option is invalid\n");
                                break;
                }
        }
}


#include <stdio.h>
#define SIZE 50

int queue[SIZE];
```

```
int front = -1, rear = -1;


void enqueue(int x) {
   if (rear == SIZE - 1) {
      printf("Queue is full\n");
      return;
   }
   if (front == -1) front = 0;
   queue[++rear] = x;
}


int dequeue() {
   if (front == -1 || front > rear) {
      printf("Queue is empty\n");
      return -1;
   }
   return queue[front++];
}
void interleave(int n){
          int half = n / 2;
   int aux[SIZE];
   int i;
   for (i = 0; i < half; i++) {
      aux[i] = dequeue();
   }
   for (i = 0; i < half; i++) {
      enqueue(aux[i]);
      enqueue(dequeue());
   }
```

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

int main() {
   int n, x,i;
   printf("Enter even number of elements: ");
   scanf("%d", &n);

   printf("Enter elements:\n");
   for (i = 0; i < n; i++) {
      scanf("%d", &x);
      enqueue(x);
   }

   interleave(n);

   printf("Interleaved Queue:\n");
   display();
```

```
   return 0;
}

#include<stdio.h>
#define n 10
int stack[n];
int top=-1;
int queue[n];
int front=-1,rear=-1;
void enqueue(){
        if(rear==n-1){
                printf("Queue is full!\n");
                return;
        }
        else{
                int el;
                printf("Enter the element:");
                scanf("%d",&el);
                if(front==-1){
                        front=0;
                }
                rear++;
                queue[rear]=el;
        }
}
void dequeue(){
        if(front==-1 || front>rear){
                printf("Queue is empty!\n");
                return;
```

```
        }
        else{
                printf("%d is dequeued from queue\n",queue[front]);
                front++;
        }
}
void reverse(){
        if(front==-1 || front>rear){
                printf("Queue is empty!\n");
                return;
        }
        else{
                while(front!=-1 && front<=rear){
                        stack[++top]=queue[front++];
                }
                front=0,rear=-1;
                while(top!=-1)
                        queue[++rear]=stack[top--];
                }
        }
}
void display(){
        if(front==-1 || front>rear){
                printf("Queue is empty!\n");
                return;
        }
        else{
                int i;
                printf("elements in queue are:");
```

```
                for(i=front;i<=rear;i++){
                        printf("%d\t",queue[i]);
                }
        }
}
int main(){
        int ch;
        while(1){

        printf("Operations\n1.Inqueue()\n2.Dequeue()\n3.reverse()\n4.Display()\n5.Exit()
\nEnter the option:");
                scanf("%d",&ch);
                switch(ch){
                        case 1:
                                enqueue();
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                reverse();
                                printf("Reversed queue is:");
                                display();
                                break;
                        case 4:
                                display();
                                break;
                        case 5:
                                printf("Exitingg...\n");
                                return;
```

```
                default:
                        printf("The option is invalid\n");
                        break;
                }
        }
}

#include <stdio.h>
#define SIZE 50
int s1[SIZE], s2[SIZE];
int top1 = -1, top2 = -1;
void push1(int x) { s1[++top1] = x; }
int pop1() { return s1[top1--]; }
void push2(int x) { s2[++top2] = x; }
int pop2() { return s2[top2--]; }

void enqueue(int x) {
  push1(x);
}

int dequeue() {
  if (top1 == -1 && top2 == -1) {
    printf("Queue is empty\n");
    return -1;
  }
  if (top2 == -1) {
    while (top1 != -1) {
      push2(pop1());
    }
```

```
    }
    return pop2();
}

void display() {
    if (top1 == -1 && top2 == -1) {
        printf("Queue is empty\n");
        return;
    }
    int i;
    printf("Queue elements: ");
    for ( i = top2; i >= 0; i--) printf("%d ", s2[i]);
    for ( i = 0; i <= top1; i++) printf("%d ", s1[i]);
    printf("\n");
}

int main() {
    int choice, value;
    while (1) {
        printf("\nOperations:\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\nEnter choice:
");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                value = dequeue();
```

```
            if (value != -1) printf("%d dequeued\n", value);
            break;
        case 3:
            display();
            break;
        case 4:
            return 0;
        default:
            printf("Invalid choice\n");
      }
   }
}
```

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *next;
};
struct node *newnode,*head,*temp;
void create(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
```

```
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=NULL;
                if(head==NULL){
                        head=temp=newnode;
                }
                else{
                        temp->next=newnode;
                        temp=newnode;
                }
        }
}
void Ibegin(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=head;
                head=newnode;
        }
}
void Iend(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
```

```
        else{
                temp=head;
                while(temp->next!=NULL){
                        temp=temp->next;
                }
                printf("Enter data:");
                scanf("%d",&newnode->data);
                temp->next=newnode;
                temp=newnode;
                newnode->next=NULL;
        }
}
void Ipos(){
        int length=0,pos,i=1;
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                temp=head;
                while(temp!=NULL){
                        length++;
                        temp=temp->next;
                }
                printf("Enter position you want to insert:");
                scanf("%d",&pos);
                if(pos>length){
                        printf("Position is invalid!!\n");
                }
```

```
                else{
                        temp=head;
                        while(i<pos-1){
                                temp=temp->next;
                                i++;
                        }
                        printf("Enter data:");
                        scanf("%d",&newnode->data);
                        newnode->next=temp->next;
                        temp->next=newnode;

                }
}
}
void display(){
        temp=head;
        if(head==NULL){
                printf("List is empty!\n");
        }
        else{
                temp=head;
                while(temp!=NULL){
                        printf("%d\t",temp->data);
                        temp=temp->next;
                }
        }
}
void search(){
        int search,flag=0,i=0;
```

```
        printf("Enter the element you want to search:");
        scanf("%d",&search);
        temp=head;
        while(temp!=NULL){
                if(temp->data==search){
                        flag=1;
                        break;
                }
                i++;
                temp=temp->next;
        }
        if(flag==1){
                printf("%d is present at %d node\n",search,i+1);
        }
        else if(flag==0){
                printf("%d is not present in linked list\n",search);
        }
}
int main(){
        int ch;
        while(1){

        printf("Operations:\n1.Create()\n2.Insert_begin()\n3.Insert_end()\n4.Insert_pos()
\n5.Display()\n6.Search()\n7.Exit()\nEnter the option:");
                scanf("%d",&ch);
                switch(ch){
                        case 1:
                                create();
                                break;
                        case 2:
```

```
                    lbegin();
                    break;
                case 3:
                    lend();
                    break;
                case 4:
                    lpos();
                    break;
                case 5:
                    display();
                    break;
                case 6:
                    search();
                    break;
                case 7:
                    printf("Exiting....\n");
                    return 0;
                default:
                    printf("Operration is invalid!\n");
                    break;
        }
}
}
```

```
#include<stdio.h>
#include<stdlib.h>
struct node{
```

```
        int data;
        struct node *next;
};
struct node *newnode,*head,*temp;
void create(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=NULL;
                if(head==NULL){
                        head=temp=newnode;
                }
                else{
                        temp->next=newnode;
                        temp=newnode;
                }
        }
}
void display(){
        temp=head;
        if(head==NULL){
                printf("List is empty!\n");
        }
        else{
                temp=head;
```

```
                while(temp!=NULL){
                        printf("%d\t",temp->data);
                        temp=temp->next;
                }
        }
}
void Dbegin(){
        if(head==NULL){
                printf("Linked list is empty!\n");
        }
        else{
                temp=head;
                printf("%d is deleted\n",head->data);
                head=head->next;
                free(temp);
        }
}
void Dend(){
        if(head==NULL){
                printf("Linked list is empty!\n");
        }
        else{
                struct node *previous;
                temp=head;
                while(temp->next!=NULL){
                        previous=temp;
                        temp=temp->next;
                }
                if(temp==head){
```

```
                head=NULL;
            }
            else{
                previous->next=NULL;
            }
            free(temp);
        }
    }
    void Dpos(){
        if(head==NULL){
            printf("Linked list is empty!\n");
        }
        else{
            int pos,length=0,i=1;
            temp=head;
            while(temp!=NULL){
                length++;
                temp=temp->next;
            }
            printf("Enter position you want to delete:");
            scanf("%d",&pos);
            if(pos>length){
                printf("Position is invalid!!\n");
            }
            else if(pos==1){
                Dbegin();
            }
            else{
                struct node *nextnode;
```

```
                temp=head;
                while(i<pos-1){
                        temp=temp->next;
                        i++;
                }
                nextnode=temp->next;
                temp->next=nextnode->next;
                free(nextnode);
        }
    }
}
int main(){
    int ch;
    while(1){

    printf("Operations:\n1.Create()\n2.Insert_begin()\n3.Insert_end()\n4.Insert_pos()
\n5.Display()\n6.Exit()\nEnter the option:");
        scanf("%d",&ch);
        switch(ch){
            case 1:
                create();
                break;
            case 2:
                Dbegin();
                break;
            case 3:
                Dend();
                break;
            case 4:
                Dpos();
```

```
                        break;
                case 5:
                        display();
                        break;
                case 6:
                        printf("Exiting....\n");
                        return 0;
                default:
                        printf("Operration is invalid!\n");
                        break;
        }
}
}


#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *next;
};
struct node *newnode,*head,*temp;
void create(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
```

```
                scanf("%d",&newnode->data);
                newnode->next=NULL;
                if(head==NULL){
                        head=temp=newnode;
                }
                else{
                        temp->next=newnode;
                        temp=newnode;
                }
        }
}
void lbegin(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=head;
                head=newnode;
        }
}
void lend(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
```

```
                temp=head;
                while(temp->next!=NULL){
                        temp=temp->next;
                }
                printf("Enter data:");
                scanf("%d",&newnode->data);
                temp->next=newnode;
                temp=newnode;
                newnode->next=NULL;
        }
}
void lpos(){
        int length=0,pos,i=1;
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                temp=head;
                while(temp!=NULL){
                        length++;
                        temp=temp->next;
                }
                printf("Enter position you want to insert:");
                scanf("%d",&pos);
                if(pos>length){
                        printf("Position is invalid!!\n");
                }
                else{
```

```
                temp=head;
                while(i<pos-1){
                        temp=temp->next;
                        i++;
                }
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=temp->next;
                temp->next=newnode;

        }
}
}
void display(){
        temp=head;
        if(head==NULL){
                printf("List is empty!\n");
        }
        else{
                temp=head;
                while(temp!=NULL){
                        printf("%d\t",temp->data);
                        temp=temp->next;
                }
        }
}
void reverse(){
        temp=head;
        if(head==NULL){
```

```
            printf("List is empty!\n");
        }
        else{
            struct node *prev=NULL,*next;
            while(temp!=NULL){
                next=temp->next;
                temp->next=prev;
                prev=temp;
                temp=next;
            }
            head=prev;
        }
}
int main(){
        int ch;
        while(1){

        printf("Operations:\n1.Create()\n2.Insert_begin()\n3.Insert_end()\n4.Insert_pos()
\n5.Display()\n6.Reverse()\n7.Exit()\nEnter the option:");
            scanf("%d",&ch);
            switch(ch){
                case 1:
                    create();
                    break;
                case 2:
                    Ibegin();
                    break;
                case 3:
                    Iend();
                    break;
```

```
                        case 4:
                                lpos();
                                break;
                        case 5:
                                display();
                                break;
                        case 6:
                                reverse();
                                break;
                        case 7:
                                printf("Exiting....\n");
                                return 0;
                        default:
                                printf("Operration is invalid!\n");
                                break;
        }
}
}


#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *next;
};
struct node *newnode,*head,*temp;
void create(){
        newnode=(struct node*)malloc(sizeof(struct node));
```

```
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=NULL;
                if(head==NULL){
                        head=temp=newnode;
                }
                else{
                        temp->next=newnode;
                        temp=newnode;
                }
        }
}
void Ibegin(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=head;
                head=newnode;
        }
}
void Iend(){
```

```
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                temp=head;
                while(temp->next!=NULL){
                        temp=temp->next;
                }
                printf("Enter data:");
                scanf("%d",&newnode->data);
                temp->next=newnode;
                temp=newnode;
                newnode->next=NULL;
        }
}
void Ipos(){
        int length=0,pos,i=1;
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                temp=head;
                while(temp!=NULL){
                        length++;
                        temp=temp->next;
                }
                printf("Enter position you want to insert:");
```

```
                scanf("%d",&pos);
                if(pos>length){
                        printf("Position is invalid!!\n");
                }
                else{
                        temp=head;
                        while(i<pos-1){
                                temp=temp->next;
                                i++;
                        }
                        printf("Enter data:");
                        scanf("%d",&newnode->data);
                        newnode->next=temp->next;
                        temp->next=newnode;

                }
}
}
void display(){
        temp=head;
        if(head==NULL){
                printf("List is empty!\n");
        }
        else{
                temp=head;
                while(temp!=NULL){
                        printf("%d\t",temp->data);
                        temp=temp->next;
                }
```

```
            }
    }
    void bubble() {
      if (head == NULL) {
        printf("List is empty!\n");
        return;
      }

      int swapped;
      struct node *temp;

      do {
        swapped = 0;
        temp = head;
        int el;

        while (temp->next != NULL) {
          if (temp->data > temp->next->data) {
            el = temp->data;
            temp->data = temp->next->data;
            temp->next->data = el;
            swapped = 1;
          }
          temp = temp->next;
        }
      } while (swapped);
    }
```

```
int main(){
	int ch;
	while(1){

	printf("Operations:\n1.Create()\n2.Insert_begin()\n3.Insert_end()\n4.Insert_pos()
\n5.Display()\n6.Reverse()\n7.Exit()\nEnter the option:");
		scanf("%d",&ch);
		switch(ch){
			case 1:
				create();
				break;
			case 2:
				Ibegin();
				break;
			case 3:
				Iend();
				break;
			case 4:
				Ipos();
				break;
			case 5:
				display();
				break;
			case 6:
				bubble();
				break;
			case 7:
				printf("Exiting....\n");
				return 0;
			default:
```

```
                    printf("Operration is invalid!\n");
                    break;
            }
    }
    }


#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node *next;
};
struct node *newnode,*head,*temp;
void create(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=NULL;
                if(head==NULL){
                        head=temp=newnode;
                }
                else{
                        temp->next=newnode;
                        temp=newnode;
```

```
                }
        }
}
void lbegin(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                newnode->next=head;
                head=newnode;
        }
}
void lend(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                temp=head;
                while(temp->next!=NULL){
                        temp=temp->next;
                }
                printf("Enter data:");
                scanf("%d",&newnode->data);
                temp->next=newnode;
                temp=newnode;
```

```
                newnode->next=NULL;
        }
}
void lpos(){
        int length=0,pos,i=1;
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                temp=head;
                while(temp!=NULL){
                        length++;
                        temp=temp->next;
                }
                printf("Enter position you want to insert:");
                scanf("%d",&pos);
                if(pos>length){
                        printf("Position is invalid!!\n");
                }
                else{
                        temp=head;
                        while(i<pos-1){
                                temp=temp->next;
                                i++;
                        }
                        printf("Enter data:");
                        scanf("%d",&newnode->data);
                        newnode->next=temp->next;
```

```
                temp->next=newnode;


            }
    }
    }
    void display(){
        temp=head;
        if(head==NULL){
            printf("List is empty!\n");
        }
        else{
            temp=head;
            while(temp!=NULL){
                printf("%d\t",temp->data);
                temp=temp->next;
            }
        }
    }
    void findmid(){
        temp=head;
        if(head==NULL){
            printf("List is empty!\n");
        }
        else{
            int length=0,i;
            temp=head;
            while(temp!=NULL){
                length++;
                temp=temp->next;
```

```
                }
                temp=head;

                        for(i=0;i<length/2;i++){
                                temp=temp->next;
                        }
                        printf("%d is mid element\n",temp->data);

        }
}
void findnth(){
        temp=head;
        if(head==NULL){
                printf("List is empty!\n");
        }
        else{
                int find,i=1;
                printf("enter the nth from end you want to find:");
                scanf("%d",&find);
                int length=0;
                temp=head;
                while(temp!=NULL){
                        length++;
                        temp=temp->next;
                }
                if(find>length){
                        printf("Nth position is invalid\n");
                }
                else{
```

```
                temp=head;
                while(i<=length-find){
                        temp=temp->next;
                        i++;
                }
                printf("%d is %d element from end\n",temp->data,find);
        }
    }
}
int main(){
    int ch;
    while(1){

    printf("Operations:\n1.Create()\n2.Insert_begin()\n3.Insert_end()\n4.Insert_pos()
\n5.Display()\n6.Find_mid()\n7.Find_nth()\n8.Exit()\nEnter the option:");
        scanf("%d",&ch);
        switch(ch){
            case 1:
                create();
                break;
            case 2:
                lbegin();
                break;
            case 3:
                lend();
                break;
            case 4:
                lpos();
                break;
            case 5:
```

```
                display();
                break;
            case 6:
                findmid();
                break;
            case 7:
                findnth();
                break;
            case 8:
                printf("Exiting....\n");
                return 0;
            default:
                printf("Operration is invalid!\n");
                break;
        }
}
}
```

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *next;
    struct node *prev;
};
struct node *newnode,*head,*tail,*temp;
void create(){
```

```
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                if(head==NULL){
                        head=tail=newnode;
                        newnode->next=NULL;

                        newnode->prev=NULL;
                }
                else{
                        tail->next=newnode;
                        tail=newnode;
                        newnode->prev = tail;
                        tail->next=NULL;
                }
        }
}
void Ibegin(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
```

```
                if (head==NULL){
                        create();
                }
                else{
                        temp=head;
                        newnode->next=head;
                        head=newnode;
                        newnode->prev=NULL;
                }
        }
}
void lend(){
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                if (head==NULL){
                        create();
                }
                else{
                tail->next = newnode;
   newnode->prev = tail;
   tail = newnode;
        newnode->next=NULL;
                }
        }
```

```
}
void Ipos(){

        int length=0,pos,i=1;
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode==NULL){
                printf("Memory Allocation failed!\n");
        }
        else{
                printf("Enter data:");
                scanf("%d",&newnode->data);
                temp=head;
                while(temp!=NULL){
                        length++;
                        temp=temp->next;
                }
                printf("Enter position you want to insert:");
                scanf("%d",&pos);
                if(pos>length){
                        printf("Position is invalid!!\n");
                }
                else{
                        temp=head;
                        while(i<pos-1){
                                temp=temp->next;
                                i++;
                        }
                        newnode->next=temp->next;
                        newnode->prev=temp;
```

```
                        temp->next=newnode;


                }
        }
}
void Dbegin(){
        if(head==NULL){
                printf("Linked list is empty!\n");
        }
        else{
                temp=head;
                head=temp->next;
                temp->prev=NULL;
                free(temp);
        }
}
void Dend(){
        if(head==NULL){
                printf("Linked list is empty!\n");
        }
        else if (head == tail) {  // Only one node present
    free(head);
    head = tail = NULL;
  }
        else {
    temp = tail;
    tail = tail->prev;
    tail->next = NULL;
    free(temp);
```

```
    }
}
void Dpos(){
        if(head==NULL){
                printf("Linked list is empty!\n");
        }
}
void display(){
        temp=head;
        if(head==NULL){
                printf("List is empty!\n");
        }
        else{
                temp=head;
                while(temp!=NULL){
                        printf("%d\t",temp->data);
                        temp=temp->next;
                }
        }
}
int main(){
        int ch;
        while(1){

        printf("Operations:\n1.Create()\n2.Insert_begin()\n3.Insert_end()\n4.Insert_pos()
\n5.Delete_begin()\n6.Delete_end()\n7.Delete_pos()\n8.Display()\n9.Exit()\nEnter the
option:");
                scanf("%d",&ch);
                switch(ch){
                        case 1:
```

```
            create();
            break;
        case 2:
            Ibegin();
            break;
        case 3:
            Iend();
            break;
        case 4:
            Ipos();
            break;
        case 5:
            Dbegin();
            break;
        case 6:
            Dend();
            break;
        case 7:
            Dpos();
            break;
        case 8:
            display();
            break;
        case 9:
            printf("Exiting...\n");
            return 0;
        default:
            printf("Invalid option!\n");
            break;} }}
```