DATA STRUCTURES USING C++
&
ELECTRICAL AND ELECTRONICS

LABORATORY MANUAL

B.TECH
(II YEAR – I SEM)
(2017-18)

DEPARTMENT OF
INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

DATA STRUCTURES USING C++

LABORATORY MANUAL

B.TECH
(II YEAR – I SEM)
(2017-18)

DEPARTMENT OF
INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

DEPARTMENT OF INFORMATION TECHNOLOGY

Vision

- To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

1. To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

2. To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

3. To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science and Engineering, the graduates will have the following Program Specific Outcomes:

1. **Fundamentals and critical knowledge of the Computer System:-** Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2. **The comprehensive and Applicative knowledge of Software Development:** Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3. **Applications of Computing Domain & Research:** Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.
12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

OBJECTIVES AND OUTCOMES

Objectives:

- To write and execute programs in C++ to solve problems using data structures such as arrays, linked lists, stacks, queues, trees, graphs, hash tables and search trees.
- To write and execute write programs in C++ to implement various sorting and searching methods.

Outcomes:

- Ability to identify the appropriate data structure for given problem.
- Graduate able to design and analyze the time and space complexity of algorithm or program.
- Ability to effectively use compilers includes library functions, debuggers and trouble shooting.

Vision:

To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission:

To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers.

Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

RECOMMENDED SYSTEM / SOFTWARE REQUIREMENTS:

1. Intel based desktop PC of 166MHz or faster processor with at least 64 MB RAM and 100 MB free disk space.
2. turbo C++ compiler or GCC compilers

USEFUL TEXT BOOKS / REFERECES:

1. Data structures, Algorithms and Applications in C++, S.Sahni, University Press (India) Pvt.Ltd, 2nd edition, Universities Press Orient Longman Pvt. Ltd.
2. Data structures and Algorithms in C++, Michael T.Goodrich, R.Tamassia and .Mount, Wiley student edition, John Wiley and Sons.
3. Data structures using C and C++, Langsam, Augenstein and Tanenbaum, PHI.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF INFORMATION TECHNOLOGY

DATA STRUCTURES USING C++ Lab Manual

List of programs

S.No	Name of the program	Page no	Date	Faculty sign
1.	Write a C++ programs to implement recursive and non recursive i) Linear search ii) Binary search	**1**		
2.	Write a C++ programs to implement i) Bubble sort ii) Selection sort iii) quick sort iv) insertion sort	**6**		
3.	Write a C++ programs to implement the following using an array. a) Stack ADT b) Queue ADT	**12**		
4.	Write a C++ programs to implement list ADT to perform following operations a) Insert an element into a list. b) Delete an element from list c) Search for a key element in list d)count number of nodes in list	**18**		
5.	Write C++ programs to implement the following using a singly linked list. Stack ADT b) Queue ADT	**26**		
6	Write C++ programs to implement the deque (double ended queue) ADT using a doubly linked list and an array.	**32**		
7	Write a C++ program to perform the following operations: a) Insert an element into a binary search tree. b) Delete an element from a binary search tree. c) Search for a key element in a binary search tree.	**38**		
8	Write C++ programs for implementing the following sorting methods: Merge sort b) Heap sort	**43**		
9	.Write C++ programs that use recursive functions to traverse the given binary tree in a) Preorder b) in order and c) post order.	**48**		
10	Write a C++ program to perform the following operations a) Insertion into a B-tree b) Deletion from a B-tree	**54**		
11	Write a C++ program to perform the following operations a)Insertion into an AVL-tree b) Deletion from an AVL-tree	**62**		
12	Write a C++ program to implement all the functions of a dictionary (ADT)	**70**		

Week 1: write a C++ programs to implement recursive and non recursive i) Linear search ii) Binary

Aim: To implement Linear search and binary search recursively and non recursively

Description:

i) LINEAR SEARCH (SEQUENTIAL SEARCH): Search begins by comparing the first element of the list with the target element. If it matches, the search ends. Otherwise, move to next element and compare. In this way, the target element is compared with all the elements until a match occurs. If the match do not occur and there are no more elements to be compared, conclude that target element is absent in the list.

For example consider the following list of elements.

5 9 7 8 11 2 6 4

To search for element 11(i.e Key element = 11). first compare the target element with first element in list i.e. 5. Since both are not matching we move on the next elements in the list and compare. Finally found the match after 5 comparisons.

Algorithm for Linear search

```
Linear_Search (A[ ], N, val , pos )
Step 1 : Set pos = -1 and k = 0
Step 2 : Repeat  while k < N
            Begin
Step 3 :   if A[ k ] = val
                  Set pos = k
                  print pos
                  Goto step 5
            End while
Step 4 : print "Value is not present"
Step 5 : Exit
```

Source code: Non recursive C++ program for Linear search

```
#include<iostream>
using namespace std;
int Lsearch(int list[ ],int n,int key);
int main()
{
int  n,i,key,list[25],pos;
        cout<<"enter no of elements\n";
        cin>>n;
        cout<<"enter "<<n<<" elements ";
        for(i=0;i<n;i++)
                cin>>list[i];
        cout<<"enter key to search";
        cin>>key;
        pos= Lsearch (list,n,key);
        if(pos==-1)
                cout<<"\nelement not found";
        else
                cout<<"\n element found at index "<<pos;
}
```

```
/*function for linear search*/
int Lsearch(int list[],int n,int key)
{
int i,pos=-1;
        for(i=0;i<n;i++)
        if(key==list[i])
        {
                pos=i;
                break;
        }
return pos;
}
```

Results

Source code: Recursive C++ program for Linear search

```
#include<iostream>
using namespace std;
int Rec_Lsearch(int list[ ],int n,int key);
int main()
{
int  n,i,key,list[25],pos;
        cout<<"enter no of elements\n";
        cin>>n;
        cout<<"enter "<<n<<" elements ";
        for(i=0;i<n;i++)
                cin>>list[i];
        cout<<"enter key to search";
        cin>>key;
        pos=Rec_Lsearch(list,n,key);
        if(pos==-1)
                cout<<"\nelement not found";
        else
        cout<<"\n element found at index "<<pos;
}
/*recursive function for linear search*/
int Rec_Lsearch(int list[],int n,int key)
{
if(n<=0)
   return -1;
if(list[n]==key)
   return n;
   else
   return Rec_Lsearch(list,n-1,key);
}
```

Results

ii) Binary Searching: Before searching, the list of items should be sorted in ascending order. First compare the key value with the item in the mid position of the array. If there is a match, we can return immediately the position. if the value is less than the element in middle location of the array, the required value is lie in the lower half of the array.if the value is greater than the element in middle location of the array, the required value is lie in the upper half of the array. We repeat the above procedure on the lower half or upper half of the array.

Algorithm:

```
Binary_Search (A [ ], U_bound, VAL)
Step 1 : set BEG = 0 , END = U_bound , POS = -1
Step 2 : Repeat while (BEG <= END )
Step 3 :      set MID = ( BEG + END ) / 2
Step 4 :      if  A [ MID ] == VAL then
                  POS = MID
                  print  VAL " is available at ", POS
                  GoTo Step 6
                End if
                if A [ MID ] > VAL then
                  set END = MID – 1
                 Else
                  set BEG = MID + 1
                 End if
           End while
Step 5 :   if POS = -1 then
                print  VAL " is not present "
           End if
Step 6 : EXIT
```

Source code: Non recursive C++ program for binary search

```
#include<iostream>
using namespace std;
int binary_search(int list[],int key,int low,int high);
int main()
{
int n,i,key,list[25],pos;
        cout<<"enter no of elements\n" ;
        cin>>n;
```

```
            cout<<"enter "<<n<<" elements in ascending order ";
            for(i=0;i<n;i++)
            cin>>list[i];
            cout<<"enter key to search" ;
            cin>>key;
            pos=binary_search(list,key,0,n-1);
            if(pos==-1)
                        cout<<"element not found" ;
            else
                        cout<<"element found at index "<<pos;
}
/*  function for binary search*/
 int binary_search(int list[],int key,int low,int high)
{
int mid,pos=-1;
            while(low<=high)
            {
                        mid=(low+high)/2;
                        if(key==list[mid])
                        {
                                    pos=mid;
                                    break;
                        }
                        else  if(key<list[mid])
                                    high=mid-1;
                        else
                                    low=mid+1;
             }
            return pos;
}
```

Results

Source code: Recursive C++ program for binary search

```
#include<iostream>
using namespace std;
int rbinary_search(int list[],int key,int low,int high);
int main()
{
int n,i,key,list[25],pos;
            cout<<"enter no of elements\n" ;
            cin>>n;
            cout<<"enter "<<n<<" elements in ascending order ";
            for(i=0;i<n;i++)
```

```
        cin>>list[i];
        cout<<"enter key to search" ;
        cin>>key;
        pos=rbinary_search(list,key,0,n-1);
        if(pos==-1)
                cout<<"element not found" ;
        else
                cout<<"element found at index "<<pos;
}
 /*recursive function for binary search*/
int rbinary_search(int list[ ],int key,int low,int high)
{
int mid,pos=-1;
        if(low<=high)
        {
                mid=(low+high)/2;
                if(key==list[mid])
                {
                        pos=mid;
                          return pos;
                }
                else  if(key<list[mid])
                      return rbinary_search(list,key,low,mid-1);
                else
                        return rbinary_search(list,key,mid+1,high);
         }
        return pos;
}
```

Results

Assignment :-

Task	date	sign	Remarks
1.Write a program to find an element in the list of elements using linear and binary search non recursively ,provide a provision to select between linear and binary searching.			
2. Write a program to find an element in the list of elements using linear and binary search recursively, provide a provision to select between linear and binary searching.			
3. write a program to implement searching using linked list			
4.Submit an analysis report of merits and demerits of linear and binary searching.			

Week 2: write a C++ programs to implement i) Bubble sort ii) Selection sort iii) quick sort iv) insertion

Aim: To implement i) Bubble sort ii) Selection sort iii) Quick sort iv) Insertion sort
Description:

i)Bubble sort

The bubble sort is an example of exchange sort. In this method, repetitive comparison is performed among elements and essential swapping of elements is done. Bubble sort is commonly used in sorting algorithms. It is easy to understand but time consuming i.e. takes more number of comparisons to sort a list . In this type, two successive elements are compared and swapping is done. Thus, step-by-step entire array elements are checked. It is different from the selection sort. Instead of searching the minimum element and then applying swapping, two records are swapped instantly upon noticing that they
are not in order.

ALGORITHM:

Bubble_Sort (A [ ] , N)

```
Step 1: Start
Step 2: Take an array of n elements
Step 3: for i=0,………….n-2
Step 4: for j=i+1,…….n-1
Step 5: if  arr[j]>arr[j+1]    then
             Interchange arr[j] and arr[j+1]
            End of if
Step 6: Print the sorted array arr
 Step 7:Stop
```

Source code: Write a program to sort a list of numbers using bubble sort

```
#include<iostream>
using namespace std;
void bubble_sort(int list[30],int n);
int main()
{
int n,i;
int  list[30];
        cout<<"enter no of elements\n";
        cin>>n;
        cout<<"enter "<<n<<" numbers ";
        for(i=0;i<n;i++)
        cin>>list[i];
        bubble_sort (list,n);
        cout<<"  after sorting\n";
        for(i=0;i<n;i++)
        cout<<list[i]<<endl;
return 0;
}
void bubble_sort (int list[30],int n)
{
int temp ;
int i,j;
        for(i=0;i<n;i++)
        for(j=0;j<n-1;j++)
        if(list[j]>list[j+1])
        {
                temp=list[j];
                list[j]=list[j+1];
                list[j+1]=temp;
        }
}
```

Results

ii) Selection sort (Select the smallest and Exchange):

The first item is compared with the remaining n-1 items, and whichever of all is lowest, is put in the first position. Then the second item from the list is taken and compared with the remaining (n-2) items, if an item with a value less than that of the second item is found on the (n-2) items, it is swapped (Interchanged) with the second item of the list and so on.

Algorithm:

```
Selection_Sort ( A [ ] , N )
    Step 1 :start
    Step 2: Repeat  For  K  =  0  to N –  2
                Begin
    Step 3 :    Set  POS = K
    Step 4 :    Repeat for J = K + 1 to N –  1
                 Begin
                     If A[ J ] < A [ POS ]
                                  Set  POS  =  J
                 End  For
    Step 5 :    Swap  A [ K ]  with A [ POS ]
                End For
    Step 6 : stop
```

Source code: Program to implement selection sort

```
#include<iostream>
using namespace std;
void selection_sort (int list[],int n);
int main()
{
int n,i;
int  list[30];
```

```
        cout<<"enter no of elements\n";
        cin>>n;
        cout<<"enter "<<n<<" numbers ";
        for(i=0;i<n;i++)
        cin>>list[i];
        selection_sort (list,n);
        cout<<"  after sorting\n";
        for(i=0;i<n;i++)
        cout<<list[i]<<endl;
return 0;
}
void selection_sort (int list[],int n)
{
int min,temp,i,j;
        for(i=0;i<n;i++)
        {
                min=i;
                for(j=i+1;j<n;j++)
                {
                        if(list[j]<list[min])
                        min=j;
                }
                temp=list[i];
                list[i]=list[min];
                list[min]=temp;
        }
}
```

Results

iii) Quick sort: It is a divide and conquer algorithm. Quick sort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quick sort can then recursively sort the sub-arrays.

ALGORITHM:
Step 1: Pick an element, called a pivot, from the array.
Step 2: Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.

Step 3: Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

Source code: program to implement Quick sort

```
#include<iostream>
using namespace std;
void quicksort(int x[],int Lb,int Ub)
{
int down,up,pivot,t;
        if(Lb<Ub)
        {
        down=Lb;
        up=Ub;
        pivot=down;
        while(down<up)
        {
                while((x[down]<=x[pivot])&&(down<Ub))down++;
                while(x[up]>x[pivot])up--;
                if(down<up)
                {
                        t=x[down];
                        x[down]=x[up];
                        x[up]=t;
                }/*endif*/
          }
            t=x[pivot];
            x[pivot]=x[up];
            x[up]=t;
         quicksort( x,Lb,up-1);
         quicksort( x,up+1,Ub);
         }
}
int main()
{
int n,i;
int  list[30];
        cout<<"enter no of elements\n";
        cin>>n;
        cout<<"enter "<<n<<" numbers ";
        for(i=0;i<n;i++)
        cin>>list[i];
        quicksort(list,0,n-1);
        cout<<"  after sorting\n";
        for(i=0;i<n;i++)
        cout<<list[i]<<endl;
return 0;
}
```

Results

iv) Insertion sort: It iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain

ALGORITHM:

```
Step 1: start
Step 2: for i ← 1 to length(A)
Step 3:   j ← i
Step 4:   while j > 0 and A[j-1] > A[j]
Step 5:       swap A[j] and A[j-1]
Step 6:       j ← j - 1
Step 7:   end while
Step 8: end for
Step9:  stop
```

Source code: program to implement insertion sort

```
#include<iostream>
using namespace std;
void insertion_sort(int a[],int n)
{
int i,t,pos;
        for(i=0;i<n;i++)
        {
                t=a[i];
                pos=i;
                while(pos>0&&a[pos-1]>t)
                {
                        a[pos]=a[pos-1];
                        pos--;
                }
                a[pos]=t;
        }
}
int main()
{
```

```
int n,i;
int  list[30];
          cout<<"enter no of elements\n";
          cin>>n;
          cout<<"enter "<<n<<" numbers ";
          for(i=0;i<n;i++)
          cin>>list[i];
          insertion_sort(list,n);
          cout<<"  after sorting\n";
          for(i=0;i<n;i++)
          cout<<list[i]<<endl;
return 0;
}
```

Results

Assignment:-

Task	date	sign	Remarks
1.What are the various time complexities of different sorting algorithms			
2.Write a program to implement all above sorting techniques in a single program. provide a menu for selection of various sorting techniques during runtime.			
3.Write a program to implement above sorting technique using dynamic memory allocation			
4.write a program to count no of operations performed in each step in all sorting algorithms			

Week 3: Write C++ programs to implement the following using an array. Stack ADT b) Queue ADT

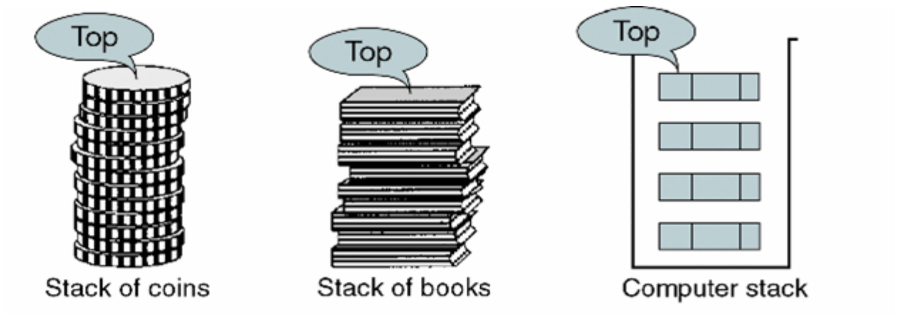
Aim: To implement Stack ADT and Queue ADT using an array

Description:

Stack:It is an ordered collection of data elements into which new elements may be inserted and from which elements may be deleted at one end called the "TOP" of stack.

- A stack is a last-in-first-out (LIFO) structure.
- Insertion operation is referred as "PUSH" and deletion operation is referred as "POP".
- The most accessible element in the stack is the element at the position "TOP".
- Stack must be created as empty.
- Whenever an element is pushed into stack, it must be checked whether the stack is full or not.
- Whenever an element is popped form stack, it must be checked whether the stack is empty or not.

We can implement the stack ADT either with array or linked list.



ALGORITHM: push()

```
Step 1: if top> =max-1 then
Step 2: Display the stack overflows
Step 3: else then
Step 4: top ++
Step 5: assign stack[top]=x
Step 6: Display element is inserted
```

ALGORITHM pop()

```
Step 1: if top = =-1 then
Step 2: Display the stack is underflows
Step 3: else
Step 4: assign x=stack[top]
Step 5:  top- -
Step 6:  return x
```

Source code: To implement Stack ADT using an array

```
#include<iostream>
using namespace std;
#include<stdlib.h>
#define max 50
template <class T>
```

```
class stack
{
private:
            T top,stk[50],item;
public:
                        stack();
            void push();
            void pop();
            void display();
};
template <class T>
stack<T>::stack()
{
top=-1;
}
//code to push an item into stack;
template <class T>
void stack<T>::push()
{
if(top==max-1)
            cout<<"Stack Overflow...\n";
else
    {
    cout<<"Enter an item to be pushed:";
    top++;
    cin>>item;
    stk[top]=item;
    cout<<"Pushed Sucesfully....\n";
    }
}
template <class T>
void stack<T>::pop()
{
if(top==-1)
            cout<<"Stack is Underflow";
else
            {
            item=stk[top];
            top--;
            cout<<item<<" is poped Sucesfully....\n";
            }
}
template <class T>
void stack<T>::display()
{
            if(top==-1)
                        cout<<"Stack Under Flow";
            else
            {
                        for(int i=top;i>-1;i--)
                        {
                                cout<<"|"<<stk[i]<<"|\n";
                                cout<<"----\n";
                        }
                }
        }
        int main()
        {
        int choice;
        stack<int>st;
                while(1)
                {
                cout<<"\n\n*****Menu for Skack operations*****\n\n";
                cout<<"1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n";
                cout<<"Enter Choice:";
                cin>>choice;
                switch(choice)
                {
                case 1:
                        st.push();
                        break;
                case 2:
                        st.pop();
                        break;
                case 3: cout<<"Elements in the Stack are....\n";
                        st.display();
                        break;
                case 4:
                        exit(0);
                default:cout<<"Invalid choice...Try again...\n";
                }
          }
        }
```

Results

QUEUE

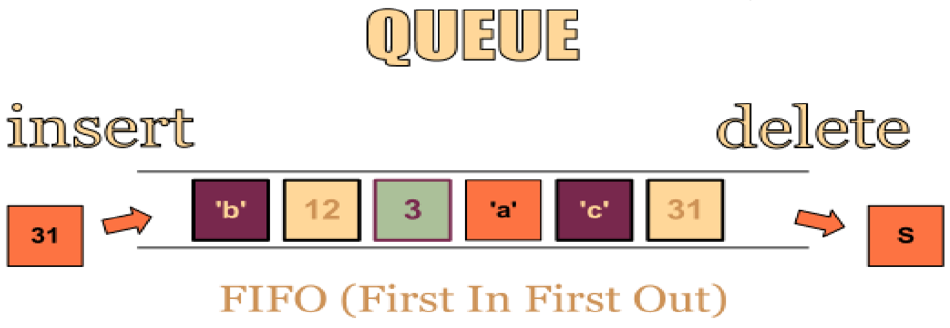
DESCRIPTION:

Queue is a data structure in which the elements are added at one end, called the rear, and deleted from the other end, called the front. A First In First Out data structure (FIFO).The rear of the queue is accessed whenever a new element is added to the queue, and the front of the queue is accessed whenever an element is deleted from the queue. As in a stack, the middle elements in the queue are in accessible, even if the queue elements are sorted in an array.

BASIC QUEUE OPERATIONS:

1. initializeQueue(): Initializes the queue to an empty state.
2. Determines whether the queue is empty. If the queue is empty, it returns the value true; otherwise, it returns the value false.
3. Determines whether the queue is full. If the queue is empty, it returns the value true; otherwise, it returns the value false.
4. rear: Returns the last element of the queue. Prior to this operation, the queue must exit.
5. front: Returns the front, that is, the first element of the queue. Priority to this operation, the queue must exit.

Queue can be stored either in an array or in linked list. We will consider both implementations. Because elements are added at one end and remove from the other end, we need two pointers to keep track of the front and rear of the queue, called queueFront and queueRear. Queues are restricted versions of arrays and linked lists. The middle terms of queue should not be accessed directly.



Source code: To implement Queue ADT using an array

```
#include<stdlib.h>
#include<iostream>
using namespace std;
#define max 5
template <class T>
class queue
{
private:T q[max],item;
        int front,rear;
public: queue();
        void insert_q();
        void delete_q();
        void display_q();
};
```

```
template <class T>
queue<T>::queue()
{
        front=rear=-1;
}

//code to insert an item into queue;
template <class T>
void queue<T> ::insert_q()
{
if(rear>=max-1)
        cout<<"queue Overflow...\n";
else
        {
                if(front>rear)
                  front=rear=-1;
                else
                 {                if(front==-1)
                                        front=0;
                        rear++;
                cout<<"Enter an item to be inserted:";
                        cin>>item;
                        q[rear]=item;
                        cout<<"inserted Sucesfully..into queue..\n";
                }
        }
}
template <class T>
void queue<T>::delete_q()
{
if(front==-1||front>rear)
        {
                front=rear=-1;
                cout<<"queue is Empty....\n";
        }
else
        {
                item=q[front];
                front++;
                cout<<item<<" is deleted Sucesfully....\n";
        }
}
template <class T>
void queue<T>::display_q()
{
         if(front==-1||front>rear)
        {
                front=rear=-1;
                cout<<"queue is Empty....\n";
        }
```

```
            else
            {
                    for(int i=front;i<=rear;i++)
                    cout<<"|"<<q[i]<<"|<--";
            }
}
int main()
{
int choice;
queue<int> q;
                while(1)
                        {
                                cout<<"\n\n*****Menu for QUEUE operations*****\n\n";
                                cout<<"1.INSERT\n2.DELETE\n3.DISPLAY\n4.EXIT\n";
                                cout<<"Enter Choice:";
                                cin>>choice;
                                switch(choice)
                                {
                                        case 1:         q.insert_q();
                                                        break;
                                        case 2:         q.delete_q();
                                                        break;
                                        case 3:         cout<<"Elements in the queue are....\n";
                                                        q.display_q();
                                                        break;
                                        case 4:         exit(0);
                                        default:        cout<<"Invalid choice...Try again...\n";
                                }

                        }
            return 0;
}
```

Results

Assignment :-

Task	Date	Sign	remark
1.Write a program to perform matching of parenthesis using stack.			
2.Write a program to perform evaluation of postfix expression			
3.Write a program to convert given infix expression to post fix			

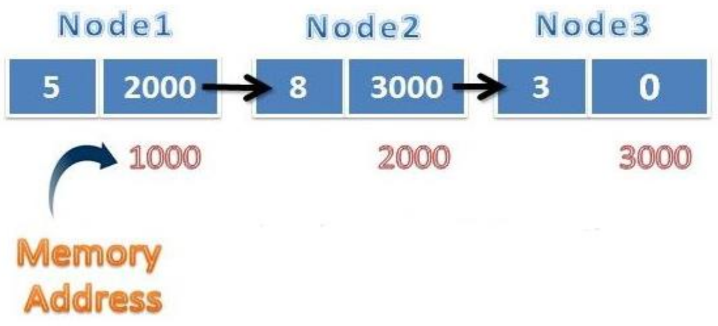
Week 4: C++ programs to implement list ADT to perform following operations
a) Insert an element into a list. b) Delete an element from list
c) Search for a key element in list d)count number of nodes in list

Aim: To implement list ADT to perform following operations
a) Insert an element into a list. b) Delete an element from list
c) Search for a key element in list d)count number of nodes in list

Description:

List ADT

A **linked list** is a data structure consisting of a group of nodes which together represent a sequence. Each node is composed of a data part and a reference (in other words, a *link*) to the next node in the sequence.



The Linked List is a collection of elements called nodes, each node of which stores two items of information, i.e., data part and link field.

The data part of each node consists the data record of an entity.

The link field is a pointer and contains the address of next node.

The beginning of the linked list is stored in a pointer termed as head which points to the first node.

The head pointer will be passed as a parameter to any method, to perform an operation.

First node contains a pointer to second node, second node contains a pointer to the third node and so on.

The last node in the list has its next field set to NULL to mark the end of the list.

- There are several variants of linked lists. These are as follows:
    - Singly linked list
    - Circular linked list
    - Doubly linked list
    - Doubly circular linked list

SINGLE LINKED LIST:

Single Linked List is a collection of nodes. Each node contains 2 fields: I) info where the information is stored and ii) link which points to the next node in the list.

The node is like this:

Node

Info (or) data	Link (or) next

The operations that can be performed on single linked lists includes: insertion, deletion and traversing the list.
Various operations on a single linked list are
1.Insertion of a node into list
2.Deletion of a node from list
3.Traversal of the list

Source code: To Implement LIST ADT in C++

```
#include<stdlib.h>
#include<iostream.h>
#include<conio.h>
class node
{
public:
        int data;
        node *next;
};
class List
{
        int item;
        node *head;
public: List( );
        void insert_front( );
        void insert_end( );
        void delete_front( );
        void delete_end( );
        void display( );
        int node_count();
        void delete_before_pos();
        void delete_after_pos();
};
List::List( )
{
        head=NULL;
}
//code to insert an item at front List;
void List::insert_front( )
{
 node *p;
   cout<<"Enter an element to be inserted:";
   cin>>item;
   p=new node;
   p->data=item;
   p->next=NULL;
   if(head==NULL)
   {
        head=p;
   }
   else
```

```
          { p->next=head;
            head=p;
     }
     cout<<"\nInserted at front of  Linked List  Sucesfully....\n";
 }


//code to insert an item at end List
void List::insert_end( )
{
 node *p;
     cout<<"Enter an element to be inserted:";
     cin>>item;
     p=new node;
     p->data=item;
     p->next=NULL;
     if(head==NULL)
     {
          head=p;
     }
     else
          {
          node*t;
          t=head;
          while(t->next!=NULL)
          t=t->next;
          t->next=p;
          }
     cout<<"\nInserted an element at end of Linked List  Sucesfully....\n";
 }

void List::delete_front( )
{
node*t;
if(head==NULL)
          cout<<"\nList is Underflow";
else
   {      item=head->data;
          t=head;
          head=head->next;
          cout<<"\n"<<item<<" is deleted Sucesfully from List....\n";
          delete(t);
   }
}

void List::delete_end( )
{
node*t,*prev;
          if(head==NULL)
                    cout<<"\nList is Underflow";
```

```
        else
        {
                t=head;
                if(head->next==NULL)
                {
                        cout<<"\n"<<t->data<<" is deleted Sucesfully from List....\n";
                        delete(t);
                        head=NULL;
                }
                else
                {
                        while(t->next!=NULL)
                        {
                                prev=t;
                                t=t->next;
                        }
                        prev->next=NULL;
                        cout<<"\n"<<t->data<<" is deleted Sucesfully from List....\n";
                        delete(t);
                }
        }
}
//Delete a node  before a position
void List::delete_before_pos( )
{
int i=1;
int pos;
node*t,*prev;
        if(head==NULL)
                cout<<"\nList is Underflow";
        else
        {       cout<<"Enter position at which node has to be deleted:";
                cin>>pos;
                t=head;
                int nc=node_count();
                if(pos>nc||pos<=0)
                        cout<<"invalid position ...try again\n";
                 else
                 {
                   cout<<"Before Deletion elements in the List are..\n";
                   display();

                        while(i<pos)
                        {
                                prev=t;
                                t=t->next;
                                i++;
                        }
                        if(i==1)
                        {
```

```
                                        cout<<"\n"<<t->data<<" is deleted Sucesfully from List....\n";
                                        if(head->next==NULL)
                                                head=NULL;
                                        else
                                        {
                                                t=head;
                                            head=head->next;
                                          cout<<"\n"<<t->data<<"is deleted Sucesfully from List....\n";
                                           delete(t);
                                         }
                                }
                                else
                                {
                                        prev->next=t->next;
                                        cout<<"\n"<<t->data<<" is deleted Sucesfully from List....\n";
                                        delete(t);
                                }
                        cout<<"After Deletion elements in the List are..\n";
                        display();
                        }
    }
}
//Delete a node after a position
void List::delete_after_pos( )
{
int i=1;
int pos;
node*t,*prev;
        if(head==NULL)
                cout<<"\nList is Underflow";
        else
        {   cout<<"Enter position at which node has to be deleted:";
                cin>>pos;
                t=head;
                int nc=node_count();
                if(pos>nc||pos<=0)
                        cout<<"invalid position ...try again\n";
                 else
                 {
                  cout<<"Before Deletion elements in the List are..\n";
                   display();
                        while(i<pos)
                        {
                                prev=t;
                                t=t->next;
                                i++;
                        }
                        if(i==1)
                        {
                                cout<<"\n"<<t->data<<" is deleted Sucesfully from List....\n";
```

```
                                    if(head->next==NULL)
                                           head=NULL;
                                    else
                                    {
                                       t=head;
                                        head=head->next;
                                     cout<<"\n"<<t->data<<" is deleted Sucesfully from List....\n";
                                        delete(t);
                                     }
                              }
                              else
                              {
                                    prev->next=t->next;
                                    cout<<"\n"<<t->data<<" is deleted Sucesfully from List....\n";
                                    delete(t);
                              }
                        cout<<"After Deletion elements in the List are..\n";
                        display();
                        }
  }
}

void List::display()
{
node*t;
      if(head==NULL)
            cout<<"\nList Under Flow";
      else
      {
            cout<<"\nElements in the List are....\n";
            t=head;
        while(t!=NULL)
            {
             cout<<"|"<<t->data<<"|->";
             t=t->next;
            }
      }
}
//code to count no of nodes
int List::node_count( )
{
int nc=0;
node*t;
      if(head==NULL)
        {
            cout<<"\nList Under Flow"<<endl;
      //   cout<<"No Nodes in the Linked List are: "<<nc<<endl;
        }
      else
      {
              t=head;
             while(t!=NULL)
                    {
                     nc++;
                     t=t->next;
                    }
       //   cout<<"No Nodes in the Linked List are: "<<nc<<endl;
            }
return nc;
}
int main( )
{
int choice;
List LL;
          while(1)
          {
          cout<<"\n\n***Menu for Linked List operations***\n\n";
          cout<<"1.Insert Front\n2.Insert end\n3.Delete front\n4.Delete End\n5.DISPLAY\n";
          cout<<"6.Node Count\n7.Del before a position\n8.Del after position\n";
          cout<<"9.Clear Scrn\n10.Exit\nEnter Choice:";
          cin>>choice;
          switch(choice)
          {
          case 1: LL.insert_front( );
                    break;
          case 2: LL.insert_end( );
                    break;
          case 3: LL.delete_front( );
                    break;
          case 4: LL.delete_end( );
                    break;
          case 5: LL.display( );
                    break;
          case 6:cout<<"No of nodes in List:"<<LL.node_count();
                    break;
          case 7:LL.delete_before_pos();
                    break;
          case 8:LL.delete_after_pos();
                    break;
          case 9:clrscr();
                break;
          case 10:exit(0);
          default:cout<<"Invalid choice...Try again...\n";
          }
  }
}
```

Results

Assignment:-

Task	Date	Sign	Remark
1.Write a program to concatenate two linked lists			
2.Write a program to reverse a given linked list			
3.Write a program to generate two lists from a given linked list such that first list contains all elements in odd places and second list consists of all elements in even places			

Week 5: Write C++ programs to implement the following using a singly linked list.
a) Stack ADT b) Queue ADT

Aim: To implement Stack ADT and Queue ADT using a singly linked list.

Description:

A Stack is a collection of items in which new items may be deleted at end to implement stack using linked list we need to define a node which in turn consist of data a pointer to the next node. The advantage of representing stack using linked lists is that we can decide which end should be top of a stack. And since the array size is fixed, in the array (linear) representation of stack, only fixed number of elements can be pushed onto the stack. If in a program the number of elements to be pushed exceeds the size of the array, the program may terminate in an error. We must overcome these problems.

By using linked lists we can dynamically organize data (such as an ordered list).Therefore , ;ogically the stack is never full. The stack is full only if we run out of memory space. In the below program we select front end as top if stack in which we cab add or remove data.

Source code: To implement Stack ADT using a singly linked list.

```
#include<stdlib.h>
#include<iostream>
using namespace std;
template <class T>
class node
{
public:
        T data;
        node<T>*next;
};

template <class T>
class stack
{
private:
          T item;
          node<T> *top;
public: stack();
                  void push();
                  void pop();
                  void display();
};
template <class T>
stack<T>::stack()
{
top=NULL;
}
//code to push an item into stack;
template <class T>
```

```
void stack<T>::push()
{
 node<T>*t;
 node<T>*p;
    cout<<"Enter an item to be pushed:";
    cin>>item;
       p=new node<T>;
      p->data=item;
      p->next=top;
      top=p;
      cout<<"\nPushed Sucesfully....\n";
 }

template <class T>
void stack<T>::pop()
{
node<T>*t;
if(top==NULL)
            cout<<"\nStack is Underflow";
else
          {
            item=top->data;
            top=top->next;
            cout<<"\n"<<item<<" is poped Sucesfully....\n";
            }

}
template <class T>
void stack<T>::display()
{
node<T>*t;
            if(top==NULL)
                        cout<<"\nStack Under Flow";
            else
            {      cout<<"\nElements in the Stack are....\n";
                        t=top;
                   while(t!=NULL)
                        {
                                    cout<<"|"<<t->data<<"|\n";
                                    cout<<"----\n";
                                    t=t->next;
                        }
            }
}
int main()
{
int choice;
stack<int>st;
            while(1)
            {
```

```
        cout<<"\n\n***Menu for Skack operations***\n\n";
        cout<<"1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n";
        cout<<"Enter Choice:";
        cin>>choice;
        switch(choice)
        {
        case 1:
                st.push();
                break;
        case 2:
                st.pop();
                break;
        case 3: st.display();
                break;
        case 4:
                exit(0);
        default:cout<<"Invalid choice...Try again...\n";
        }
  }
}
```

Results

Description: Queue is a data structure in which the elements are added at one end, called the **rear**, and deleted from the other end, called the **front**. A First In First Out data structure (FIFO). The rear of the queue is accessed whenever a new element is added to the queue, and the front of the queue is accessed whenever an element is deleted from the queue. As in a stack, the middle elements in the queue are in accessible, even if the queue elements are sorted in an array.

Source code: To implement QUEUE ADT using a singly linked list

```
#include<stdlib.h>
#include<iostream.h>
template <class T>
class node
{
public:
          T data;
          node<T>*next;
};
```

```
template <class T>
class queue
{
private:
                T item;
                friend class node<T>;
                node<T> *front,*rear;
public: queue();
                void insert_q();
                void delete_q();
                void display_q();
};
template <class T>
queue<T>::queue()
{
                front=rear=NULL;
}
//code to push an item into queue;
template <class T>
void queue<T>::insert_q()
{
 node<T>*p;
    cout<<"Enter an element to be inserted:";
    cin>>item;
                p=new node<T>;
    p->data=item;
    p->next=NULL;
    if(front==NULL)
    {
        rear=front=p;
    }
    else
                {
                 rear->next=p;
                        rear=p;
                }
    cout<<"\nInserted into Queue Sucesfully....\n";
 }
//code to delete an element
template <class T>
void queue<T>::delete_q()
{
node<T>*t;
if(front==NULL)
                cout<<"\nqueue is Underflow";
else
   {
                        item=front->data;
                        t=front;
                        front=front->next;
                cout<<"\n"<<item<<" is deleted Sucesfully from queue....\n";
        }
delete(t);
}
//code to display elements in queue
template <class T>
void queue<T>::display_q()
{
node<T>*t;
        if(front==NULL)
                cout<<"\nqueue Under Flow";
        else
        {
                cout<<"\nElements in the queue are....\n";
                t=front;
            while(t!=NULL)
                {
                  cout<<"|"<<t->data<<"|<-";
                  t=t->next;
                }
        }
}
int main()
{
int choice;
queue<int>q1;

        while(1)
        {
                cout<<"\n\n***Menu for Queue operations***\n\n";
                cout<<"1.Insert\n2.Delete\n3.DISPLAY\n4.EXIT\n";
                cout<<"Enter Choice:";
                cin>>choice;
                switch(choice)
                {
                        case 1: q1.insert_q();
                                break;
                        case 2: q1.delete_q();
                                break;
                        case 3: q1.display_q();
                                break;
                        case 4: exit(0);
                        default:cout<<"Invalid choice...Try again...\n";
                }
  }
  return 0;
}
```

Results

Assignment:-

Task	Date	Sign	Remark
1.Submit an analysis report on stack implemented by static allocation and dynamic allocation			
2.Submit an analysis report on Queue implemented by static allocation and dynamic allocation			
3.Submit a report on application of stack and queue with explanation			

Week 6: Write C++ programs to implement the de queue (double ended queue) ADT using a doubly linked list and an array.

Aim: To implement the de queue (double ended queue) ADT using a doubly linked list and an array.
Source code: To implement the de queue (double ended queue) ADT

```
#include <iostream.h>
#include<stdlib.h>
#include <conio.h>
template<class T>
class node
{
public:
                T data;
                node*prev;
                node*next;
};
template<class T>
class dll
{
node<T>*head;
public:
dll();
                void insert_front();
                void insert_end();
                void delete_front();
                void delete_end();
                void display();
                void insert_at_pos();
                int node_count();
};
template<class T>
 dll<T>::dll()
{
                head=NULL;
}
//code to insert node at front of list...
template<class T>
void dll<T>::insert_front()
{
node<T>*new_node;
                 int x;
                 new_node=new node<T>;
                 cout<<"Enter data into node:\n";
                 cin>>x;
     new_node->data=x;
                 new_node->prev=NULL;
                 new_node->next=NULL;
                 if(head==NULL)
                        head=new_node;
                  else
                  {
                  new_node->next=head;
                  head->prev=new_node;
                  head=new_node;
                  }
                 cout<<"Inserted node sucesfully...";
}
//code to insert node at end of list...
template<class T>
void dll<T>::insert_end()
{
node<T>*new_node;
node<T>*t;
int x;
                new_node=new node<T>;
                 cout<<"Enter data into node:\n";
                cin>>x;
                 new_node->data=x;
                 new_node->next=NULL;
                 new_node->prev=NULL;
                 if(head==NULL)
                        head=new_node;
                  else
                  {
                          t=head;
                          while(t->next!=NULL)
                                    t=t->next;
                          t->next= new_node;
                          new_node->prev=t;
                   }
                 cout<<"Inserted node sucesfully...";
}
template<class T>
void dll<T>::delete_front()
{
node<T>*temp;
                if(head==NULL)
                        cout<<"List is empty....\n ";
                else if(head->next==NULL)
                {
                temp=head;
cout<<"Deleted element from Doubly Linked List is "<<temp->data<<endl;
                        delete temp;
                        head=NULL;

                }
                else
```

```
            {
                  temp=head;
                  head=head->next;
                  head->prev=NULL;
                  cout<<"Deleted element from Doubly Linked List is "<<temp->data<<endl;
                  delete temp;
                  cout<<"Elements after deletion from Front are...\n";
                  display();
            }
}
template<class T>
void dll<T>::delete_end()
{
node<T>*t1;
node<T>*t2;
            if(head==NULL)
                  cout<<"List is empty....\n ";
            else
            {
                  t1=t2=head;
                  if(head->next==NULL)
                  {
                           head=NULL;
                  cout<<"Deleted element from Doubly Linked List is "<<t1->data<<endl;
                           delete t1;
                  }
                  else
                  {
                           while(t1->next!=NULL)
                           {
                                    t2=t1;
                                    t1=t1->next;
                           }
                           t2->next=NULL;
                  cout<<"Deleted element from Doubly Linked List is "<<t1->data<<endl;
                           delete t1;
                           cout<<"Elements after Deletion from End are...\n";
                           display();
                  }
            }
}
template<class T>
void dll<T>::insert_at_pos()
{
node<T>*new_node;
node<T>*t1;
node<T>*t2;
int x,pos,nc;
            new_node=new node<T>;
             cout<<"Enter data into node:\n";
```

```
cin>>x;
cout<<"enter Pos at which node has to be inserted:";
cin>>pos;
        new_node->data=x;
        new_node->next=NULL;
        new_node->prev=NULL;
        nc=node_count();
        cout<<"node count="<<nc<<endl;
if(pos<=0||pos>nc+1)
                  cout<<"invalid position";
else
 {
if(pos==1)
       {

                  if(head==NULL)
                             head=new_node;
                  else
                  {

                  new_node->next=head;
                  head->prev=new_node;
                  head=new_node;
                  }
       }

  else
  {
         t1=t2=head;
         int i=1;
         while(i<pos)
                  {
                             t2=t1;
                             t1=t1->next;
                             i++;
                  }

                  if(t1==NULL)
                  {
                  new_node->next=NULL;
                  }
                  else
                  {
                   t1->prev=new_node;
                  }
         t2->next= new_node;
         new_node->prev=t2;
  }
 cout<<"Inserted node sucesfully...";
}
```

```
}

template<class T>
int dll<T>:: node_count()
{
int i=0;
node<T> *t;
t=head;
while(t!=NULL)
{
t=t->next;
i++;
}
return i;
}
template<class T>
void dll<T>::display()
{
node<T>*t;
int count;
        t=head;
        if(head==NULL)
        {
            cout<<"Doubly linked list is empty.....\n";
        }
        else
        {
            cout<<"Elements in the list are........\n";
            while(t!=NULL)
             {
                cout<<"|"<<t->data<<"|-> ";
                t=t->next;
             }
        }
count=node_count();
cout<<"\n Total No of nodes in Doubly linked List are:"<<count<<endl;

}
int main()
{
dll<int> d;
int choice;
        while(1)
        {   cout<<"\n***Menu for Doubly linked list operations***\n";
            cout<<"\n1.insert front";
            cout<<"\n2.insert end";
            cout<<"\n3.delete front";
            cout<<"\n4.delete end";
            cout<<"\n5.Display";
            cout<<"\n6.insert at pos";
```

```
                        cout<<"\n7.Exit";
                        cout<<"\nEnter Choice:";
                        cin>>choice;
                        switch(choice)
                        {
                              case 1:d.insert_front();
                                          break;
                              case 2:d.insert_end();
                                          break;
                              case 3:d.delete_front();
                                          break;
                              case 4:d.delete_end();
                                          break;
                              case 5:d.display();
                                          break;
                              case 6:d.insert_at_pos();
                                          break;
                              case 7:exit(0);
                        }
                }
//return 0;
}
```

Results

Assignment:-

Task	Date	Sign	Remark
1.Implement queue using doubly linked list			
2.Implement circular Queue using doubly linked list			
3.			

Week 7: Write a C++ program to perform the following operations:

a) Insert an element into a binary search tree.
b) Delete an element from a binary search tree.
c) Search for a key element in a binary search tree.

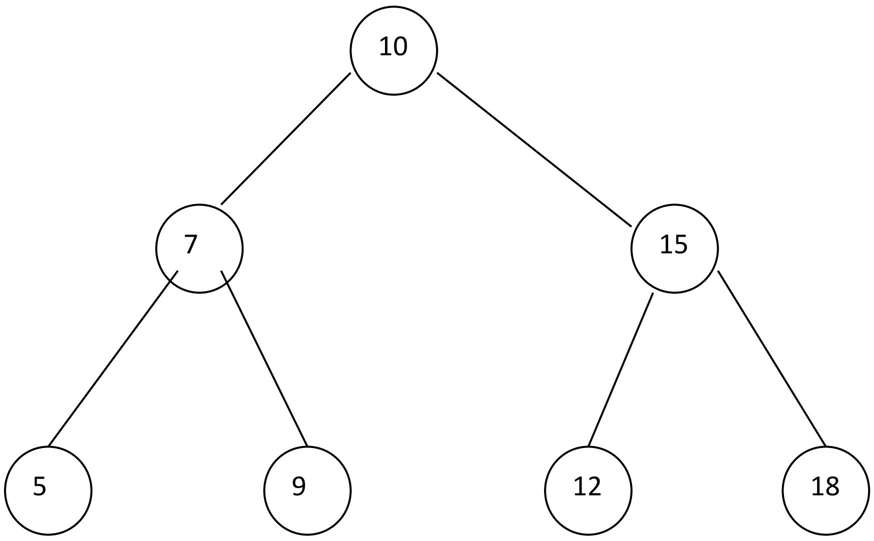
Description:

Binary Search Tree:

So to make the searching algorithm faster in a binary tree we will go for building the binary search tree. The binary search tree is based on the binary search algorithm. While creating the binary search tree the data is systematically arranged.

That means values at

left sub-tree < root node value < right sub-tree values.



Source code:

```
#include<stdlib.h>
#include<iostream.h>
class node
{
public:
            int data;
            node*lchild;
            node*rchild;
};
class bst:public node
{           int item;
            node *root;
public: bst();
            void insert_node();
            void delete_node();
            void display_bst();
            void inorder(node*);
```

```
};
bst::bst()
{
root=NULL;
}
void bst:: insert_node()
{
node *new_node,*curr,*prev;
        new_node=new node;
        cout<<"Enter data into new node";
        cin>>item;
        new_node->data=item;
        new_node->lchild=NULL;
        new_node->rchild=NULL;
        if(root==NULL)
                root=new_node;
        else
        {       curr=prev=root;
                while(curr!=NULL)
                {       if(new_node->data>curr->data)
                        {       prev=curr;
                                curr=curr->rchild;
                        }
                        else
                        {       prev=curr;
                                curr=curr->lchild;
                        }
                }
                cout<<"Prev:"<<prev->data<<endl;
                if(prev->data>new_node->data)
                        prev->lchild=new_node;
                else
                        prev->rchild=new_node;
                }
}
//code to delete a node
void bst::delete_node()
{
if(root==NULL)
        cout<<"Tree is Empty";
else
{
int key;
        cout<<"Enter the key value to be deleted";
        cin>>key;
        node* temp,*parent,*succ_parent;
        temp=root;
        while(temp!=NULL)
        {       if(temp->data==key)
                {   //deleting node with two childern
```

```
if(temp->lchild!=NULL&&temp->rchild!=NULL)
        {       //search for inorder sucessor
                node*temp_succ;
                temp_succ=temp->rchild;
                while(temp_succ->lchild!=NULL)
                {
                        succ_parent=temp_succ;
                        temp_succ=temp_succ->lchild;
                }
                temp->data=temp_succ->data;
                succ_parent->lchild=NULL;
            cout<<"Deleted sucess fully";
            return;
        }
        //deleting a node having one left child
        if(temp->lchild!=NULL&temp->rchild==NULL)
        {
        if(parent->lchild==temp)
                parent->lchild=temp->lchild;
        else
                parent->rchild=temp->lchild;
        temp=NULL;
        delete(temp);
        cout<<"Deleted sucess fully";
         return;
        }
    //deleting a node having one right child
        if(temp->lchild==NULL&temp->rchild!=NULL)
        {
        if(parent->lchild==temp)
                parent->lchild=temp->rchild;
        else
                parent->rchild=temp->rchild;
        temp=NULL;
        delete(temp);
        cout<<"Deleted sucess fully";
        return;
        }
   //deleting a node having no child
        if(temp->lchild==NULL&temp->rchild==NULL)
        {
        if(parent->lchild==temp)
                parent->lchild=NULL;
        else
                parent->rchild=NULL;
        temp=NULL;
        delete(temp);
        cout<<"Deleted sucess fully";
        return;
        }
```

```
                    }
                    else if(temp->data<key)
                          {         parent=temp;
                                    temp=temp->rchild;
                               }
                    else if(temp->data>key)
                               {        parent=temp;
                                        temp=temp->lchild;
                               }
          }//end while
  }//end if
 }//end delnode func
void bst::display_bst()
{
          if(root==NULL)
                    cout<<"\nBST Under Flow";
          else
                    inorder(root);
}
void bst::inorder(node*t)
{
          if(t!=NULL)
          {
                    inorder(t->lchild);
                    cout<<" "<<t->data;
                    inorder(t->rchild);
          }
}

int main()
{
bst bt;
int i;
          while(1)
          {
                    cout<<"****BST Operations****";
                    cout<<"\n1.Insert\n2.Display\n3.del\n4.exit\n";
                    cout<<"Enter Choice:";
                    cin>>i;
                    switch(i)
                    {
                              case 1:bt.insert_node();
                                                  break;
                              case 2:bt.display_bst();
                                                  break;
                              case 3:bt.delete_node();
                                        break;
                              case 4:exit(0);
                              default: cout<<"Enter correct choice";
                    }
```

```
            }
    }
```

Results

Assignment:-

Task	Date	Sign	Remark
1.What is a binary tree& binary search tree			
2.Applications of binary search tree			

Week 8 : Write C++ programs for implementing the following sorting methods:
a) Merge sort b) Heap sort

Aim: To implement Merge sort and Heap sort

Description:
Merge sort is an O(*n* log *n*) comparison-based sorting algorithm. It is stable, meaning that it preserves the input order of equal elements in the sorted output. It is an example of the divide and conquer algorithmic paradigm. Merge sort is so inherently sequential that it's practical to run it using slow tape drives as input and output devices. It requires very little memory, and the memory required does not change with the number of data elements. If you have four tape drives, it works as follows:

1. Divide the data to be sorted in half and put half on each of two tapes
2. Merge individual pairs of records from the two tapes; write two-record chunks alternately to each of the two output tapes
3. Merge the two-record chunks from the two output tapes into four-record chunks; write these alternately to the original two input tapes
4. Merge the four-record chunks into eight-record chunks; write these alternately to the original two output tapes
5. Repeat until you have one chunk containing all the data, sorted --- that is, for log *n* passes, where *n* is the number of records.

Conceptually, merge sort works as follows:

1. Divide the unsorted list into two sublists of about half the size
2. Divide each of the two sublists recursively until we have list sizes of length 1, in which case the list itself is returned
3. Merge the two sublists back into one sorted list.

Source code:

```
#include<iostream>
using namespace std;
  #define max 15
 template<class T>
  void merge(T a[],int l,int m,int u)
   {
  T b[max];
   int i,j,k;
            i=l; j=m+1;
             k=l;
             while((i<=m)&&(j<=u))
              {
                    if(a[i]<=a[j])
                      {
                            b[k]=a[i];
                             ++i;
                     }
                    else
                     {
                             b[k]=a[j];
                             ++j;
                    }
                     ++k;
          }
          if(i>m)
          {
                    while(j<=u)
                    {
                            b[k]=a[j];
                             ++j;
                           ++k;
                    }
          }
          else
           {
                    while(i<=m)
                    {
                            b[k]=a[i];
                            ++i;
                            ++k;
                    }
          }
          for(int r=l;r<=u;r++)
          a[r]=b[r];
}

 template <class T>
 void mergesort(T a[],int p,int q)
  {
  int mid;
  if(p<q)
   {
          mid=(p+q)/2;
         mergesort(a,p,mid);
         mergesort(a,mid+1,q);
         merge(a,p,mid,q);
         }
}


int main()
{
```

```
int n,i;
int  list[30];
            cout<<"enter no of elements\n";
            cin>>n;
            cout<<"enter "<<n<<" numbers ";
            for(i=0;i<n;i++)
            cin>>list[i];
            mergesort (list,0,n-1);
            cout<<"  after sorting\n";
            for(i=0;i<n;i++)
            cout<<list[i]<<endl;
return 0;
}
```

Results

HEAP SORT

Heap sort is a method in which a binary tree is used. In this method first the heap is created using binary tree and then heap is sorted using priority queue.

Source code:// C++ program for implementation of Heap Sort

```
#include <iostream>
using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
    int largest = i;  // Initialize largest as root
    int l = 2*i + 1;  // left = 2*i + 1
    int r = 2*i + 2;  // right = 2*i + 2

    // If left child is larger than root
    if (l < n && arr[l] > arr[largest])
        largest = l;

    // If right child is larger than largest so far
    if (r < n && arr[r] > arr[largest])
        largest = r;

    // If largest is not root
    if (largest != i)
    {
        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract an element from heap
    for (int i=n-1; i>=0; i--)
    {
        // Move current root to end
        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
    for (int i=0; i<n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

int main()
{
int n,i;
int  list[30];
          cout<<"enter no of elements\n";
          cin>>n;
          cout<<"enter "<<n<<" numbers ";
          for(i=0;i<n;i++)
          cin>>list[i];
          heapSort(list, n);
          cout << "Sorted array is \n";
    printArray(list, n);
return 0;
}
```

Results

Assignment:-

Task	Date	Sign	Remark
1.Write a program to explain time complexity in Merge sort			
2.Explain about heap sort			

Week 9 : Write C++ programs that use recursive functions to traverse the given binary tree in Preorder b) inorder and c) postorder

Aim: To implement Binary tree traversals

Description:

It is often convenient to a single list containing all the nodes in a tree. This list may correspond to an order in which the nodes should be visited when the tree is being searched. We define three such lists here, the **preorder**, **postorder** and **inorder** traversals of the tree. The definitions themselves are recursive:

- if *T* is the empty tree, then the empty list is the preorder, the inorder and the postorder traversal associated with *T*;
- if *T* = [*N*] consists of a single node, the list [*N*] is the preorder, the inorder and the postorder traversal associated with *T*;
- otherwise, *T* contains a root node *n*, and subtrees T_1,..., T_n: and
    - the *preorder* traversal of the nodes of *T* is the list containing *N*, followed, in order by the preorder traversals of T_1..., T_n;
    - the *inorder* traversal of the nodes of *T* is the list containing the inorder traversal of T_1 followed by *N* followed in order by the inorder traversal of each of T_2,..., T_n.
    - the *postorder* traversal of the nodes of *T* is the list containing in order the postorder traversal of each of T_1,..., T_n, followed by *N*.

Source code:

```
 #include<stdlib.h>
#include<iostream.h>
class node
{
public:
        int data;
        node*Lchild;
        node*Rchild;
};
class bst
{
        int item;
        node *root;
public: bst();
        void insert_node();
        void delete_node();
        void display_bst();
        void preeorder(node*);
        void inorder(node*);
        void postorder(node*);
};
```

```
bst::bst()
{
root=NULL;
}
void bst:: insert_node()
{
node *new_node,*curr,*prev;
        new_node=new node;
        cout<<"Enter data into new node";
        cin>>item;
        new_node->data=item;
        new_node->Lchild=NULL;
        new_node->Rchild=NULL;
        if(root==NULL)
                root=new_node;
        else
        {
                curr=prev=root;
                while(curr!=NULL)
                {
                        if(new_node->data>curr->data)
                        {
                                prev=curr;
                                curr=curr->Rchild;
                        }
                        else
                        {
                                prev=curr;
                                curr=curr->Lchild;
                        }
                }
                cout<<"Prev:"<<prev->data<<endl;
                if(prev->data>new_node->data)
                        prev->Lchild=new_node;
                else
                        prev->Rchild=new_node;
                }
}
//code to delete a node
void bst::delete_node()
{
if(root==NULL)
        cout<<"Tree is Empty";
else
{
        int key;
        cout<<"Enter the key value to be deleted";
        cin>>key;
        node* temp,*parent,*succ_parent;
        temp=root;
```

```
while(temp!=NULL)
{
            if(temp->data==key)
            {    //deleting node with two childern
                if(temp->Lchild!=NULL&&temp->Rchild!=NULL)
                        {           //search for sucessor
                                    node*temp_succ;
                                    temp_succ=temp->Rchild;
                                    while(temp_succ->Lchild!=NULL)
                                    {
                                                succ_parent=temp_succ;
                                                temp_succ=temp_succ->Lchild;
                                    }
                                    temp->data=temp_succ->data;
                                    succ_parent->Lchild=NULL;
                               cout<<"Deleted sucess fully";
                               return;
                        }
                        //deleting a node having one left child
                        if(temp->Lchild!=NULL&temp->Rchild==NULL)
                        {
                        if(parent->Lchild==temp)
                                    parent->Lchild=temp->Lchild;
                        else
                                    parent->Rchild=temp->Lchild;
                        temp=NULL;
                        delete(temp);
                        cout<<"Deleted sucess fully";
                         return;
                        }
                    //deleting a node having one right child
                        if(temp->Lchild==NULL&temp->Rchild!=NULL)
                        {
                        if(parent->Lchild==temp)
                                    parent->Lchild=temp->Rchild;
                        else
                                    parent->Rchild=temp->Rchild;
                        temp=NULL;
                        delete(temp);
                        cout<<"Deleted sucess fully";
                        return;
                        }
                   //deleting a node having no child
                        if(temp->Lchild==NULL&temp->Rchild==NULL)
                        {
                        if(parent->Lchild==temp)
                                    parent->Lchild=NULL;
                        else
                                    parent->Rchild=NULL;
                        temp=NULL;
                    delete(temp);
                    cout<<"Deleted sucess fully";
                    return;
                    }
                }
                else if(temp->data<key)
                   {
                        parent=temp;
                        temp=temp->Rchild;
                    }
                else if(temp->data>key)
                    {
                          parent=temp;
                          temp=temp->Lchild;
                    }
            }//end while
   }//end if
  }//end delnode func

void bst::display_bst()
{
        if(root==NULL)
                cout<<"\nBinary Search Tree is Under Flow";
        else
        {
                int ch;
                cout<<"\t\t**Binart Tree Traversals**\n";
                cout<<"\t\t1.Pree order\n\t\t2.Inorder\n\t\t3:PostOrder\n";
                cout<<"\t\tEnter Your Chice:";
                cin>>ch;
                switch(ch)
                {
                case 1: cout<<"Pree order Tree Traversal\n ";
                        preeorder(root);
                        break;
                case 2: cout<<"Inorder Tree Traversal is\n ";
                        inorder(root);
                        break;
                case 3: cout<<"Inorder Tree Traversal is\n";
                        postorder(root);
                        break;
                }
        }
}
void bst::inorder(node*t)
{
        if(t!=NULL)
        {
                inorder(t->Lchild);
                cout<<" "<<t->data;
                inorder(t->Rchild);
        }
}

void bst::preeorder(node*t)
{
        if(t!=NULL)
        {
                cout<<" "<<t->data;
                preeorder(t->Lchild);
                preeorder(t->Rchild);
        }
}
void bst::postorder(node*t)
{
        if(t!=NULL)
        {
                postorder(t->Lchild);
                postorder(t->Rchild);
                cout<<" "<<t->data;
        }
}
int main()
{
bst bt;
int i;
        while(1)
        {     cout<<"\n\n***Operations Binary Search Tree***\n";
                cout<<"1.Insert\n2.Display\n3.del\n4.exit\n";
                cout<<"Enter Choice:";
                cin>>i;
                switch(i)
                {

                        case 1:bt.insert_node();
                                        break;
                        case 2:bt.display_bst();
                                        break;
                        case 3:bt.delete_node();
                                break;
                        case 4:exit(0);

                        default:cout<<"Enter correct choice";
                }
        }
}
```

Results

Assignment:-

Task	Date	Sign	Remark
1.What is a Tree traversal.			
2.Application of Tree traversal			

Week 10 : Write a C++ program to perform the following operations
a) Insertion into a B-tree b) Deletion from a B-tree

Aim: To implement B Tree

Source code:

```
#include<iostream.h>
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 4
#define MIN 2
typedef char Type[10];
typedef struct Btree
{
Type key;
}BT;

typedef struct treenode
{
int count;
BT entry[MAX+1];
treenode *branch[MAX+1];
}node;
class B
{
node *root;
public:
int LT(char *,char *);
int EQ(char *,char *);
node *Search(Type target,node *root,int *targetpos);
int SearchNode(Type target,node *current,int *pos);
node *Insert(BT New,node *root);
int MoveDown(BT New,node *current,BT *med,node **medright);
void InsertIn(BT med,node *medright,node *current,int pos);
void Split(BT med,node *medright,node *current,int pos,BT *newmedian,node
**newright);
void Delete(Type target,node **root);
void Del_node(Type target,node *current);
void Remove(node *current,int pos);
void Successor(node *current,int pos);
void Adjust(node *current,int pos);
void MoveRight(node *current,int pos);
void MoveLeft(node *current,int pos);
 void Combine(node *current,int pos);
void InOrder(node *root);
};
```

```
int B::LT(char *a,char *b)
{
if((strcmp(a,b))<(0))
return 1;
else
return 0;
}
int B::EQ(char *a,char *b)
{
if((strcmp(a,b))==(0))
return 1;
else
return 0;
}
node* B::Search(Type target,node *root,int *targetpos)
{
if(root==NULL)
return NULL;
else if(SearchNode(target,root,targetpos))
return root;
else
return Search(target,root->branch[*targetpos],targetpos);
}

int B::SearchNode(Type target,node *current,int *pos)
{
if(LT(target,current->entry[1].key))
{
*pos=0;
return 0;
}
else
{
for(*pos=current->count;
LT(target,current->entry[*pos].key) && *pos>1;(*pos)--);
return EQ(target,current->entry[*pos].key);
}
}
node *B::Insert(BT newentry,node *root)
{
BT medentry;
node *medright;
node *New;
if(MoveDown(newentry,root, &medentry, &medright))
{
New=new node;
New->count=1;
New->entry[1]=medentry;
New->branch[0]=root;
New->branch[1]=medright;
return New;
}
return root;
}
int B::MoveDown(BT New,node *current,BT *med,node **medright)
{
int pos;
if(current==NULL)
{
*med=New;
*medright=NULL;
return 1;
}
else
{
if(SearchNode(New.key,current,&pos))
cout<<"Duplicate key\n";
if(MoveDown(New,current->branch[pos],med,medright))
if(current->count<MAX)
{
InsertIn(*med,*medright,current,pos);
return 0;
}
else
{
Split(*med,*medright,current,pos,med,medright);
return 1;
}
return 0;
}
}
void B::InsertIn(BT med,node *medright,node *current,int pos)
{
int i;
for(i=current->count;i>pos;i--)
{
current->entry[i+1]=current->entry[i];
current->branch[i+1]=current->branch[i];
}
current->entry[pos+1]=med;
current->branch[pos+1]=medright;
current->count++;
}
void B::Split(BT med,node *medright,node *current,int pos,BT *newmedian,node
**newright)
{
int i;
 int median;
 if(pos<=MIN)
            median=MIN;
 else
        median=MIN+1;
 *newright=new node;
for(i=median+1;i<=MAX;i++)
{
(*newright)->entry[i-median]=current->entry[i];
 (*newright)->branch[i-median]=current->branch[i];
}
 (*newright)->count=MAX-median;
 current->count=median;
 if(pos<=MIN)
        InsertIn(med,medright,current,pos);
 else
InsertIn(med,medright,*newright,pos-median);
 *newmedian=current->entry[current->count];
 (*newright)->branch[0]=current->branch[current->count];
 current->count--;
}

void B::Delete(Type target,node **root)
{
node *prev;
Del_node(target,*root);
if((*root)->count==0)
{
prev=*root;
*root=(*root)->branch[0];
free(prev);
}
}

void B::Del_node(Type target,node *current)
{
int pos;
if(!current)
{
        cout<<"Item not in the Btree\n";
        return;
}
else
{
if(SearchNode(target,current,&pos))
if(current->branch[pos-1])
{
Successor(current,pos);
Del_node(current->entry[pos].key,current->branch[pos]);
}
else
        Remove(current,pos);
else
```

```
        Del_node(target,current->branch[pos]);
if(current->branch[pos])
if(current->branch[pos]->count<MIN)
Adjust(current,pos);
}
}

void B::Remove(node *current,int pos)
{
int i;
for(i=pos+1;i<=current->count;i++)
{
current->entry[i-1]=current->entry[i];
current->branch[i-1]=current->branch[i];
}
current->count--;
}

void B::Successor(node *current,int pos)
{
node *leaf;
for(leaf=current->branch[pos];leaf->branch[0];
leaf=leaf->branch[0]);
current->entry[pos]=leaf->entry[1];
}

void B::Adjust(node *current,int pos)
{
if(pos==0)
if(current->branch[1]->count > MIN)
MoveLeft(current,1);
else
Combine(current,1);
else if(pos==current->count)
if(current->branch[pos-1]->count > MIN)
MoveRight(current,pos);
else
Combine(current,pos);
else if(current->branch[pos-1]->count > MIN)
MoveRight(current,pos);
else if(current->branch[pos+1]->count > MIN)
MoveLeft(current,pos+1);
else
Combine(current,pos);
}


void B::MoveRight(node *current,int pos)
{
int i;
```

```
node *t;
t=current->branch[pos];
for(i=t->count;i>0;i--)
{
t->entry[i+1]=t->entry[i];
t->branch[i+1]=t->branch[i];
}
t->branch[1]=t->branch[0];
t->count++;
t->entry[1]=current->entry[pos];
t=current->branch[pos-1];
current->entry[pos]=t->entry[t->count];
current->branch[pos]->branch[0]=t->branch[t->count];
t->count--;
}

void B::MoveLeft(node *current,int pos)
{
int c;
node *t;
        t=current->branch[pos-1];
        t->count++;
        t->entry[t->count]=current->entry[pos];
        t->branch[t->count]=current->branch[pos]->branch[0];
        t=current->branch[pos];
        current->entry[pos]=t->entry[1];
        t->branch[0]=t->branch[1];
        t->count--;
        for(c=1;c<=t->count;c++)
        t->entry[c]=t->entry[c+1];
        t->branch[c]=t->branch[c+1];
}

void B::Combine(node *current,int pos)
{
int c;
node *right;
node *left;
right=current->branch[pos];
left=current->branch[pos-1];
left->count++;
left->entry[left->count]=current->entry[pos];
left->branch[left->count]=right->branch[0];
for(c=1;c<=right->count;c++)
{
left->count++;
left->entry[left->count]=right->entry[c];
left->branch[left->count]=right->branch[c];
}
```

```
for(c=pos;c<current->count;c++)
{
current->entry[c]=current->entry[c+1];
current->branch[c]=current->branch[c+1];
}
current->count--;
free(right);
}

void B::InOrder(node *root)
{
int pos;
if(root)
{
InOrder(root->branch[0]);
for(pos=1;pos<=root->count;pos++)
{
cout<<" "<<root->entry[pos].key;
InOrder(root->branch[pos]);
}
}
}

int main()
{
int choice,targetpos;
Type inKey;
BT New;
B obj;
node *root,*target;
root=NULL;
while(1)
{
cout<<"\n IMPLEMENTATION OF B-TREE\n";
cout<<"\n1.INSERT\n2.DELETE\n3.SEARCH\n4.DISPLAY\n5.EXIT\n";
cout<<"Enter Your Choice\n";
cin>>choice;
switch(choice)
{
case 1:cout<<"enter the key to be inserted\n";
        fflush(stdin);
        gets(New.key);
        root=obj.Insert(New,root);
        break;

case 2:cout<<"enter the key to be deleted\n";
        fflush(stdin);
        gets(New.key);
        cout<<"Deleting the desired item\n";
        obj.Delete(New.key,&root);
```

```
            break;
case 3:cout<<"enter the key to be searched\n";
            fflush(stdin);
            gets(New.key);
            target=obj.Search(New.key,root,&targetpos);
            if(target)
            cout<<"The searched item"<<target->entry[targetpos].key<<endl;
            else
            cout<<"Element not found\n";
            break;

case 4:cout<<"\n InOrder Traversal\n";
            obj.InOrder(root);
            break;
case 5:exit(0);
}
}
}
```

Results

Assignment:-

Task	Date	Sign	Remark
1.What is a B-Tree			
2.Application of B-Tree			

Week 11 : Write a C++ program to perform the following operations
a) Insertion into an AVL-tree b) Deletion from an AVL-tree

Aim: To implement AVL tree

Source code:

```
# include <iostream.h>
# include <stdlib.h>
# include <conio.h>
struct node
{
  int element;
  node *left;
  node *right;
  int height;
};
typedef struct node *np;
class bstree
{
  public:
        void insert(int,np &);
        void del(int, np &);
        int deletemin(np &);
        void find(int,np &);
        np findmin(np);
        np findmax(np);
        void copy(np &,np &);
        void makeempty(np &);
        np nodecopy(np &);
        void preorder(np);
        void inorder(np);
        void postorder(np);
        int bsheight(np);
        np srl(np &);
        np drl(np &);
        np srr(np &);
        np drr(np &);
        int max(int,int);
        int nonodes(np);
};
//                Inserting a node
void bstree::insert(int x,np &p)
{
  if (p == NULL)
  {
        p = new node;
        p->element = x;
        p->left=NULL;
        p->right = NULL;
        p->height=0;
```

```
            if (p==NULL)
                        cout<<"Out of Space";
   }
   else
   {
            if (x<p->element)
            {
              insert(x,p->left);
              if ((bsheight(p->left) - bsheight(p->right))==2)
              {
                 if (x < p->left->element)
                       p=srl(p);
                 else
                       p = drl(p);
              }
            }
            else if (x>p->element)
            {
                 insert(x,p->right);
                 if ((bsheight(p->right) - bsheight(p->left))==2)
                 {
                       if (x > p->right->element)
                                   p=srr(p);
                       else
                                   p = drr(p);
                 }
            }
            else
                        cout<<"Element Exists";
            }
            int m,n,d;
            m=bsheight(p->left);
            n=bsheight(p->right);
            d=max(m,n);
            p->height = d + 1;
}
//Finding the Smallest
np bstree::findmin(np p)
{
            if (p==NULL)
            {
               cout<<"Empty Tree ";
               return p;
            }
            else
            {
              while(p->left !=NULL)
                        p=p->left;
              return p;
            }
```

```
}
//Finding the Largest
np bstree::findmax(np p)
{
            if (p==NULL)
            {
              cout<<"Empty Tree ";
              return p;
            }
            else
            {
              while(p->right !=NULL)
                  p=p->right;
              return p;
            }
}

//Finding an element
void bstree::find(int x,np &p)
{
            if (p==NULL)
              cout<<" Element not found ";
            else
            if (x < p->element)
              find(x,p->left);
            else
            if (x>p->element)
              find(x,p->right);
            else
              cout<<"       Element found !";
}
//Copy a tree
void bstree::copy(np &p,np &p1)
{
            makeempty(p1);
            p1 = nodecopy(p);
}
//                      Make a tree empty
void bstree::makeempty(np &p)
{
            np d;
            if (p != NULL)
            {
              makeempty(p->left);
              makeempty(p->right);
              d=p;
              free(d);
              p=NULL;
            }
}
```

```
//                  Copy the nodes
np bstree::nodecopy(np &p)
{
          np temp;
          if (p==NULL)
            return p;
          else
          {
            temp = new node;
            temp->element = p->element;
            temp->left = nodecopy(p->left);
            temp->right = nodecopy(p->right);
            return temp;
          }
}
//                  Deleting a node
void bstree::del(int x,np &p)
{
          np d;
          if (p==NULL)
            cout<<"Element not found ";
          else if ( x < p->element)
            del(x,p->left);
          else if (x > p->element)
            del(x,p->right);
          else if ((p->left == NULL) && (p->right == NULL))
          {
            d=p;
            free(d);
            p=NULL;
            cout<<" Element deleted !";
          }
          else if (p->left == NULL)
          {
           d=p;
           free(d);
           p=p->right;
           cout<<" Element deleted !";
          }
          else if (p->right == NULL)
          {
           d=p;
           p=p->left;
           free(d);
           cout<<" Element deleted !";
          }
          else
           p->element = deletemin(p->right);
}
```

```
int bstree::deletemin(np &p)
{
        int c;
        cout<<"inside deltemin";
        if (p->left == NULL)
        {
          c=p->element;
          p=p->right;
          return c;
        }
        else
        {
          c=deletemin(p->left);
          return c;
        }
}
void bstree::preorder(np p)
{
        if (p!=NULL)
        {
          cout<<p->element<<"-->";
          preorder(p->left);
          preorder(p->right);
        }
}
//                Inorder Printing
void bstree::inorder(np p)
{
        if (p!=NULL)
        {
          inorder(p->left);
          cout<<p->element<<"-->";
          inorder(p->right);
        }
}
//                PostOrder Printing
void bstree::postorder(np p)
{
        if (p!=NULL)
        {
          postorder(p->left);
          postorder(p->right);
          cout<<p->element<<"-->";
        }
}
int bstree::max(int value1, int value2)
{
        return ((value1 > value2) ? value1 : value2);
}
int bstree::bsheight(np p)
```

```
{
        int t;
        if (p == NULL)
                return -1;
        else
        {
                t = p->height;
                return t;
        }
}
np bstree:: srl(np &p1)
{
        np p2;
        p2 = p1->left;
        p1->left = p2->right;
        p2->right = p1;
        p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;
        p2->height = max(bsheight(p2->left),p1->height) + 1;
        return p2;
}
np bstree:: srr(np &p1)
{
        np p2;
        p2 = p1->right;
        p1->right = p2->left;
        p2->left = p1;
        p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;
        p2->height = max(p1->height,bsheight(p2->right)) + 1;
        return p2;
}
np bstree:: drl(np &p1)
{
        p1->left=srr(p1->left);
        return srl(p1);
}
np bstree::drr(np &p1)
{
        p1->right = srl(p1->right);
        return srr(p1);
}
int bstree::nonodes(np p)
{
        int count=0;
        if (p!=NULL)
        {
                nonodes(p->left);
                nonodes(p->right);
                count++;
        }
        return count;
```

```
}
int main()
{
	//clrscr();
	np root,root1,min,max;//,flag;
	int a,choice,findele,delele,lefte le,rightele,flag;
	char ch='y';
	bstree bst;
	//system("clear");
	root = NULL;
	root1=NULL;

	while(1)
	{
	cout<<"		\nAVL Tree\n";
	cout<<"			========\n";
		cout<<"1.Insertion\n2.FindMin\n";
		cout<<"3.FindMax\n4.Find\n5.Copy\n";
		cout<<"6.Delete\n7.Preorder\n8.Inorder\n";
		cout<<"9.Postorder\n10.height\n11.EXIT\n";
		cout<<"Enter the choice:";
		cin>>choice;
		switch(choice)
		{
		case 1:
			cout<<"New node's value ?";
			cin>>a;
			bst.insert(a,root);
			break;
		case 2:
			if (root !=NULL)
			{
			min=bst.findmin(root);
			cout<<"Min element : "<<min->element;
			}
			break;
		 case 3:
			if (root !=NULL)
			{
			max=bst.findmax(root);
			cout<<"Max element :		"<<max->element;
			}
			break;
		case 4:
			cout<<"Search node : ";
			cin>>findele;
			if (root != NULL)
				bst.find(findele,root);
			break;
		case 5:
```

```
                        bst.copy(root,root1);
                        bst.inorder(root1);
                        break;
                case 6:
                        cout<<"Delete Node ?";
                        cin>>delele;
                        bst.del(delele,root);
                        bst.inorder(root);
                        break;
                case 7:
                        cout<<" Preorder Printing... :";
                        bst.preorder(root);
                        break;
                case 8:
                          cout<<" Inorder Printing.... :";
                           bst.inorder(root);
                              break;
                case 9:
                              cout<<" Postorder Printing... :";
                              bst.postorder(root);
                              break;
                case 10:
                        cout<<" Height and Depth is ";
                        cout<<bst.bsheight(root);
                        //cout<<"No. of nodes:"<<bst.nonodes(root);
                        break;
                case 11:exit(0);
                }
                }
        return 0;
}
```

Results

Assignment:-

Task	Date	Sign	Remark
1. What is a height balanced tree			
2. Explain What is the necessity of rotations in AVL tree.			

Week 12 : Write a C++ program to implement all the functions of a dictionary (ADT)

Aim: To to implement all the functions of a dictionary (ADT)

Source code:

```
#include<stdlib.h>
#include<iostream.h>
class node
{
public: int key;
        int value;
        node*next;
};
class dictionary:public node
{       int k,data;
        node *head;
public: dictionary();
        void insert_d( );
        void delete_d( );
        void display_d( );
};
dictionary::dictionary( )
{       head=NULL;
}
//code to push an val into dictionary;
void dictionary::insert_d( )
{
node *p,*curr,*prev;
   cout<<"Enter an key and value to be inserted:";
   cin>>k;
   cin>>data;
   p=new node;
   p->key=k;
   p->value=data;
   p->next=NULL;
   if(head==NULL)
     head=p;
   else
   {
curr=head;
     while((curr->key<p->key)&&(curr->next!=NULL))
     { prev=curr;
       curr=curr->next;
     }
     if(curr->next==NULL)
     {
                 if(curr->key<p->key)
                       { curr->next=p;
                              prev=curr;
                 }
```

```
                        else { p->next=prev->next;
                              prev->next=p;
                        }
        }
     else
        {
p->next=prev->next;
            prev->next=p;
              }
     cout<<"\nInserted into dictionary Sucesfully....\n";
 }
}
void dictionary::delete_d( )
{
node*curr,*prev;
            cout<<"Enter key value that you want to delete...";
            cin>>k;
            if(head==NULL)
                        cout<<"\ndictionary is Underflow";
            else
              {         curr=head;
                        while(curr!=NULL)
                        {
                        if(curr->key==k)
                                    break;
                        prev=curr;
                        curr=curr->next;
                        }
             }
            if(curr==NULL)
                        cout<<"Node not found...";
            else
             {
                         if(curr==head)
                                    head=curr->next;
                         else
                                  prev->next=curr->next;
             delete curr;
             cout<<"Item deleted from dictionary...";
             }
  }
void dictionary::display_d( )
{
node*t;
            if(head==NULL)
                        cout<<"\ndictionary Under Flow";
            else
            {
                        cout<<"\nElements in the dictionary are....\n";
                        t=head;
            while(t!=NULL)
                  {
                   cout<<"<"<<t->key<<","<<t->value<<">";
                   t=t->next;
                  }
         }
}
int main( )
{
int choice;
dictionary d1;
         while(1)
         {
         cout<<"\n\n***Menu for Dictrionay operations***\n\n";
         cout<<"1.Insert\n2.Delete\n3.DISPLAY\n4.EXIT\n";
         cout<<"Enter Choice:";
         cin>>choice;
         switch(choice)
         {
         case 1: d1.insert_d();
                           break;
         case 2: d1.delete_d( );
                           break;
         case 3: d1.display_d( );
                           break;
         case 4: exit(0);
         default:cout<<"Invalid choice...Try again...\n";
         }
  }
}
```

Results

Assignment:-

Task	Date	Sign	Remark
1. Write a program to implement hash table			
2.Explain DFS and BFS			

ELECTRICAL AND ELECTRONICS

LABORATORY MANUAL

(FOR CSE & IT)

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGG

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY
(Autonomous Institution – UGC, Govt of India)
(Affiliated to JNTU, Hyderabad)
Secunderabad-14.

LIST OF EXPERIMENTS

PART- A

PART- B

PART-A

1. VERIFICATION OF KIRCHOFF'S LAWS

AIM: To verify the Kirchhoff's voltage law and Kirchhoff's current law for the given circuit.

APPARATUS REQUIRED:

S.No	Name of the equipment	Range	Type	Quantity
1	RPS	0-30V	-	1N0
2	Voltmeter	0-20 V	Digital	4 NO
3	Ammeter	0-20mA	Digital	4 NO
4	Bread board	-	-	1 NO
5	Connecting wires	-	-	Required number.
6	Resistors	470 Ω		2 NO
		1kΩ		1 NO
		680Ω		1 NO

CIRCUIT DIAGRAMS:

GIVEN CIRCUIT:

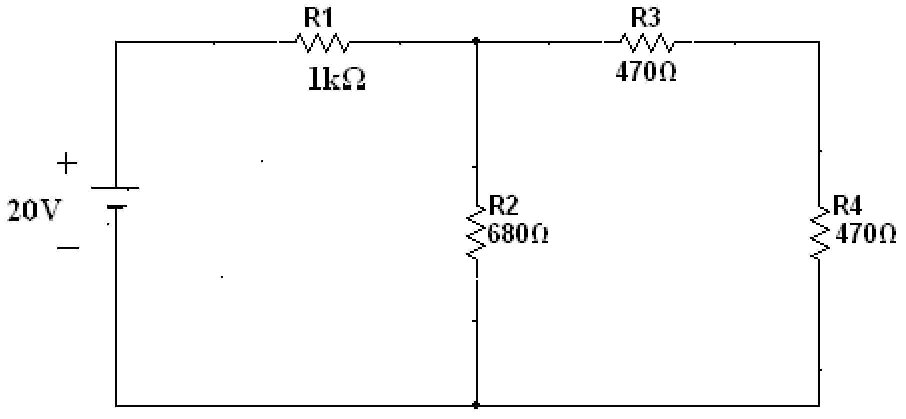


Fig (1)

1. KVL:

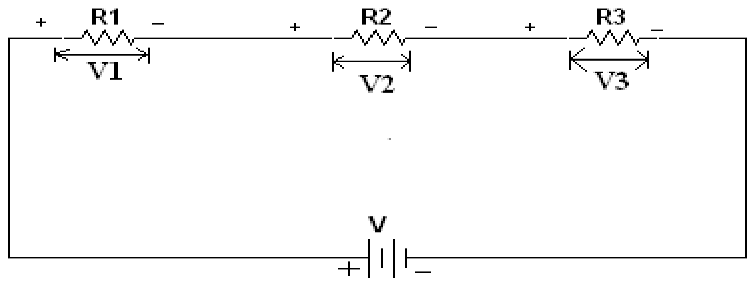
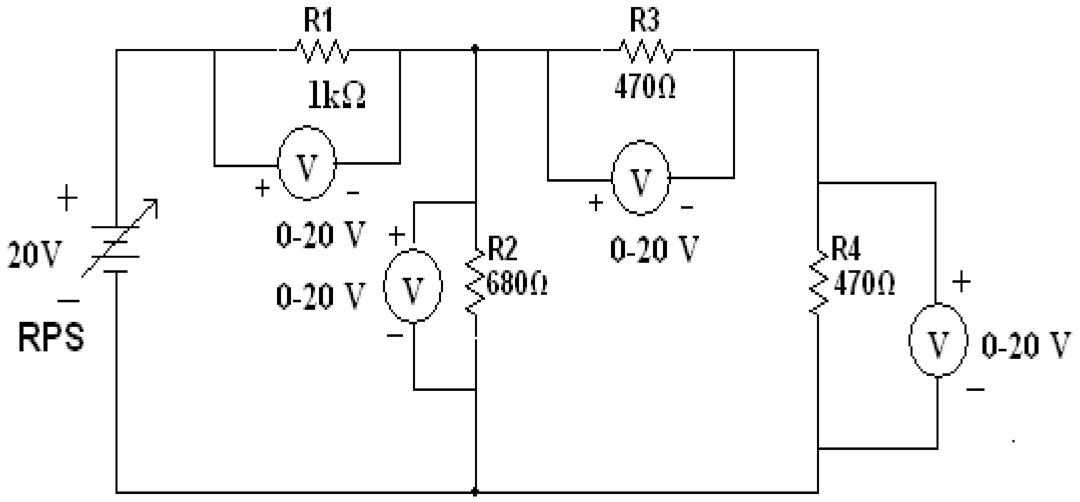


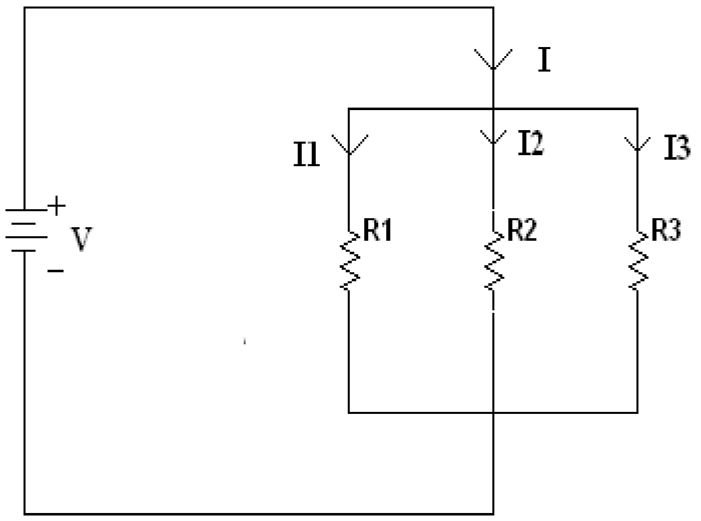
Fig (1a)

PRACTICAL CIRCUIT:



Fig(2a)

2. KCL:



Fig(1b)

PRACTICAL CIRCUIT:

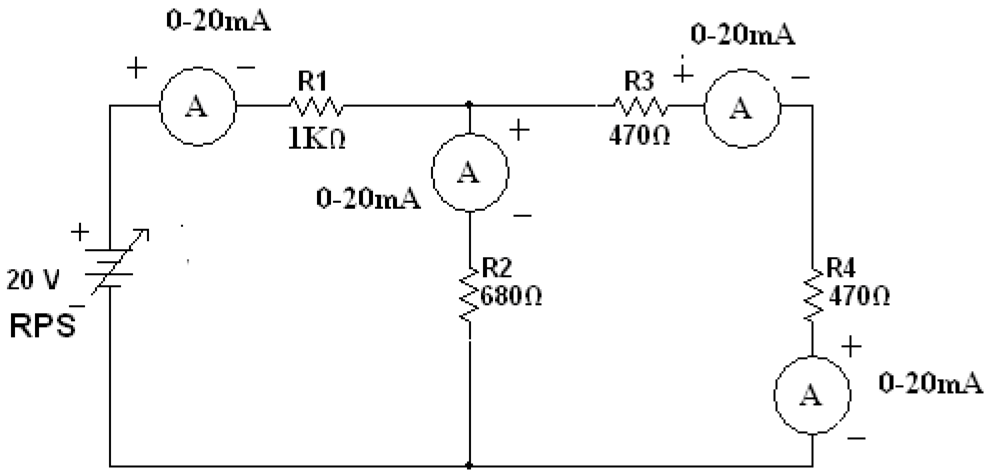


Fig (2b)

THEORY:

a) Kirchhoff’s Voltage law states that the algebraic sum of the voltage around any closed path in a given circuit is always zero. In any circuit, voltage drops across the resistors always have polarities opposite to the source polarity. When the current passes through the resistor, there is a loss in energy and therefore a voltage drop. In any element, the current flows from a higher potential to lower potential. Consider the fig (1a) shown above in which there are 3 resistors are in series. According to kickoff’s voltage law….

$$\mathbf{V = V_1 + V_2 + V_3}$$

b) Kirchhoff’s current law states that the sum of the currents entering a node equal to the sum of the currents leaving the same node. Consider the fig(1b) shown above in which there are 3 parallel paths. According to Kirchhoff’s current law...

$$\mathbf{I = I_1 + I_2 + I_3}$$

PROCEDURE:

a) Kirchhoff’s Voltage law:
   1. Connect the circuit as shown in fig (2a).
   2. Measure the voltages across the resistors.
   3. Observe that the algebraic sum of voltages in a closed loop is zero.

b) Kirchhoff’s current law:
   1. Connect the circuit as shown in fig (2b).
   2. Measure the currents through the resistors.
   3. Observe that the algebraic sum of the currents at a node is zero.

OBSERVATION TABLE:

KVL:

S.NO	VOLTAGE ACCROSS	THEORETICAL	PRACTICAL

KCL:

S.NO	CURRENT THROUGH	THEORETICAL	PRACTICAL

PRECAUTIONS:

1. Avoid loose connections.
2. Keep all the knobs in minimum position while switch on and off of the supply.

RESULT:

QUESTIONS:

1. What is another name for KCL & KVL?
2. Define network and circuit?
3. What is the property of inductor and capacitor?

2. SUPERPOSITION AND RECIPROCITY THEOREMS

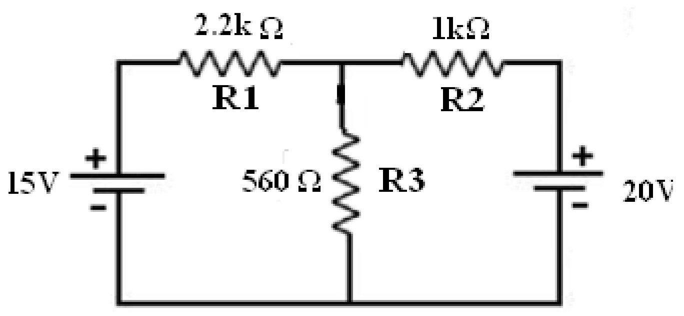
A) VERIFICATION OF SUPERPOSITION THEOREM

AIM: To verify the superposition theorem for the given circuit.

APPARATUS REQUIRED:

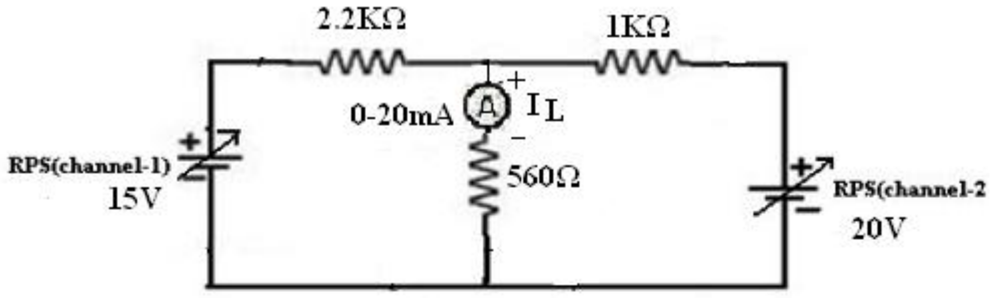
S.No	Name Of The Equipment	Range	Type	Quantity
1	Ammeter	(0-20)mA	Digital	1 NO
2	RPS	0-30V	Digital	1 NO
3	Resistors	2.2k Ω		1 NO
		1k Ω		1 NO
		560 Ω		1 NO

CIRCUIT DIAGRAM:



PRACTICAL CIRCUITS:

WhenV_1&V_2 source acting(To find I):-



Fig(1)

WhenV_1 source acting(To find I_1):-

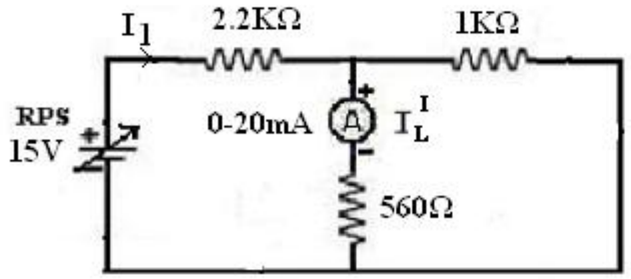


Fig (2)

WhenV2 source acting (To find I2):

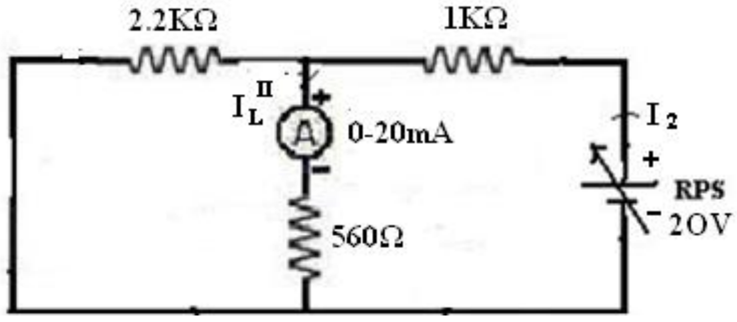


Fig (3)

THEORY:

SUPERPOSITION THEOREM:

Superposition theorem states that in a lumped ,linear, bilateral network consisting more number of sources each branch current(voltage) is the algebraic sum all currents (branch voltages), each of which is determined by considering one source at a time and removing all other sources. In removing the sources, voltage and current sources are replaced by internal resistances.

PROCEDURE:

1. Connect the circuit as per the fig (1).
2. Adjust the output voltage of sources X and Y to appropriate values (Say 15V and20V respectively).
3. Note down the current (I_L) through the 560 0hm resistor by using the ammeter.
4. Connect the circuit as per fig (2) and set the source Y (20V) to 0V.
5. Note down the current ($I_L^{1)}$ through 560ohm resistor by using ammeter.
6. Connect the circuit as per fig(3) and set the source X (15V) to 0V and source Y to 20V.
7. Note down the current (I_L^{ll}) through the 560 ohm resistor branch by using ammeter.
8. Reduce the output voltage of the sources X and Y to 0V and switch off the supply.
9. Disconnect the circuit.

THEORITICAL CALCULATIONS

From Fig(2)

$$I_1=V_1/(R_1+(R_2//R_3))$$

$$I_L^{1} = I_1 * R_2/(R_2+R_3)$$

From Fig(3)

$$I_2=V_2/(R_2+(R_1//R_3))$$

$$I_L^{11} = I_2 * R_1/(R_1+R_3)$$

$$I_L = I_L^{1} + I_L^{11}$$

TABLER COLUMNS:

From Fig(1)

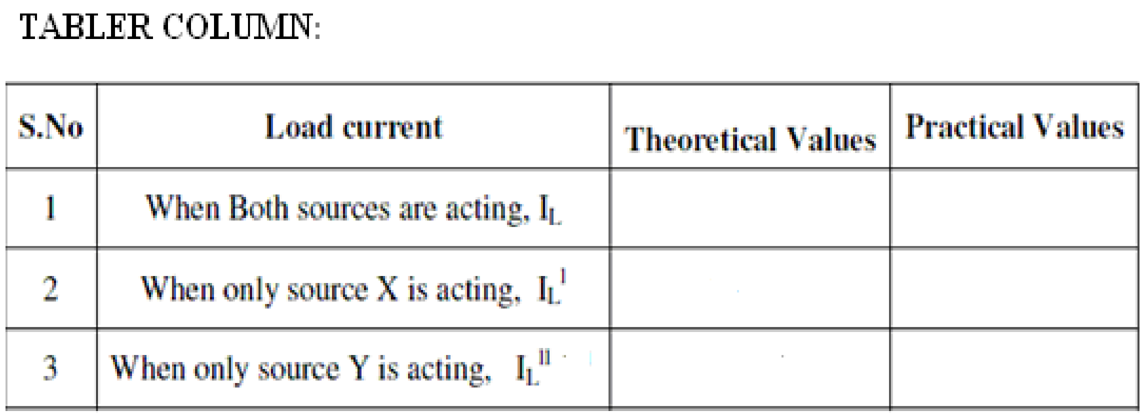
S. No	Applied voltage (V_1) Volt	Applied voltage (V_2) Volt	Current I_L (mA)

From Fig(2)

S. No	Applied voltage (V_1) Volt	Current I_L^{I} (mA)

From Fig(3)

S. No	Applied voltage (V_2) Volt	Current I_L^{II} (mA)



TABLER COLUMN:

S.No	Load current	Theoretical Values	Practical Values
1	When Both sources are acting, I_L		
2	When only source X is acting, I_L^{1}		
3	When only source Y is acting, I_L^{11}		

PRECAUTIONS:
1. Initially keep the RPS output voltage knob in zero volt position.
2. Set the ammeter pointer at zero position.
3. Take the readings without parallax error.
4. Avoid loose connections.
5. Avoid short circuit of RPS output terminals.

RESULT:

QUESTIONS:
1) What do you man by Unilateral and Bilateral network? Give the limitations of Superposition theorem?
2) What are the equivalent internal impedances for an ideal voltage source and for a Current source?
3) Transform a physical voltage source into its equivalent current source.

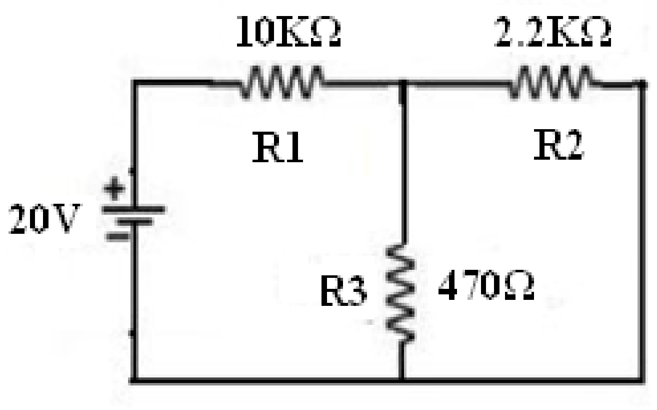
(B)RECIPROCITY THEOREM

AIM: To verify the reciprocity theorem for the given circuit.

APPARATUS REQUIRED:

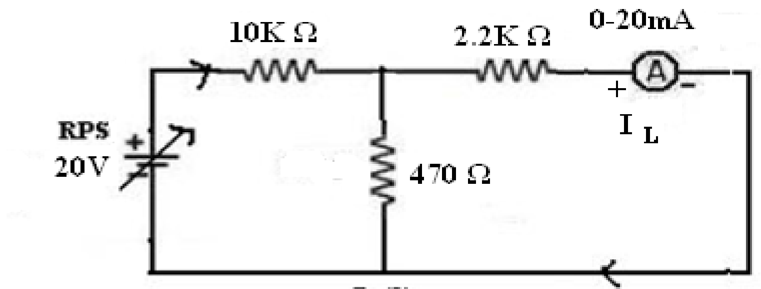
S.No	Name Of The Equipment	Range	Type	Quantity
1	Ammeter	(0-20)mA	Digital	1 NO
2	RPS	0-30V	Digital	1 NO
3	Resistors	2.2k Ω		1 NO
		10k Ω		1 NO
		470 Ω		1 NO

CIRCUIT DIAGRAM:



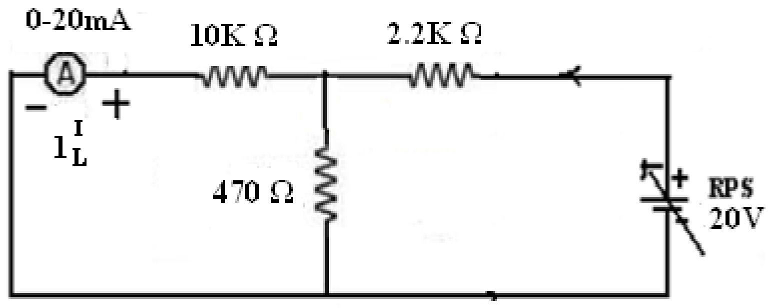
PRACTICAL CIRCUITS:

CIRCUIT – 1:



fig(1)

CIRCUIT - 2:



fig(2)

THEORY:

STATEMENT:

In any linear, bilateral, single source network, the ratio of response to the excitation is same even though the positions of excitation and response are interchanged.

PROCEDURE:

1. Connect the circuit as per the fig (1).
2. Adjust the output voltage of the regulated power supply to an appropriate value (Say 20V).
3. Note down the current through 2.2K Ω by using ammeter.

4. Reduce the output voltage of the RPS to 0V and switch-off the supply.
5. Disconnect the circuit and connect the circuit as per the fig (2).
6. Adjust the output voltage of the regulated power supply to an appropriate value (Say 20V).
7. Note down the current through 10K Ω resistor from ammeter.
8. Reduce the output voltage of the RPS to 0V and switch-off the supply.
9. Disconnect the circuit.

THEORITICAL CALCULATIONS :

From Fig(1)

$$I_1=V/(R_1+(R_2//R_3))$$

$$I_L= I_1*R_3/(R_2+R_3)$$

From Fig(2)

$$I_2=V/(R_2+(R_1//R_3))$$

$$I_L^1= I_2*R_3/(R_1+R_3)$$

TABULAR FORM:

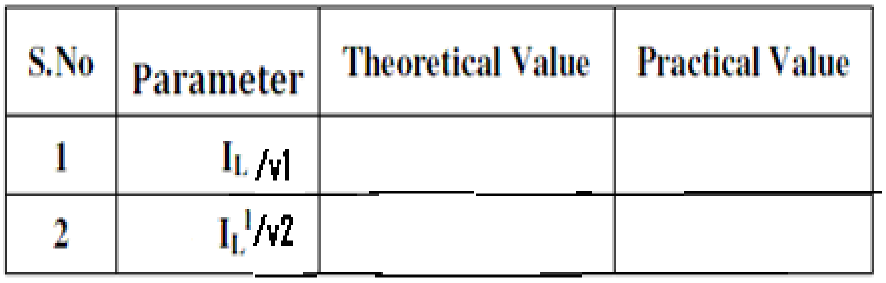
From fig 1

S. No	Applied voltage (V1) Volt	Current I_L (mA)

From fig 2

S. No	Applied voltage (V2) Volt	Current I_L^I (mA)

OBSERVATION TABLE:



S.No	Parameter	Theoretical Value	Practical Value
1	$I_L/v1$		
2	$I_L^1/v2$		

PRECAUTIONS:

1. Initially keep the RPS output voltage knob in zero volt position.
2. Set the ammeter pointer at zero position.
3. Take the readings without parallax error.
4. Avoid loose connections.
5. Avoid short circuit of RPS output terminals.
6. If voltmeter gives negative reading then interchange the terminals connections of a voltmeter

RESULT:

QUESTIONS:

1) What is reciprocity theorem?
2) Why it is not applicable for unilateral circuits.

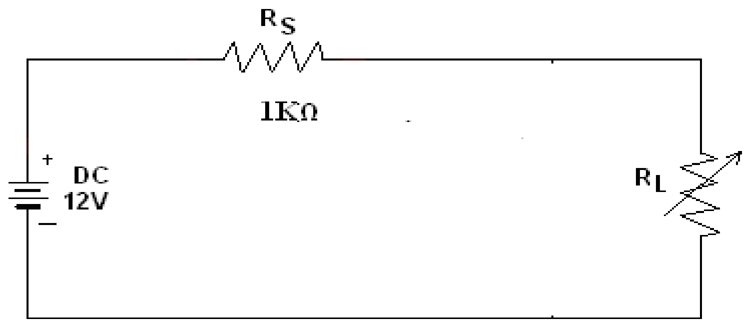
3. MAXIMUM POWER TRANSFER THEOREM

AIM: To verify the maximum power transfer theorem for the given circuit.

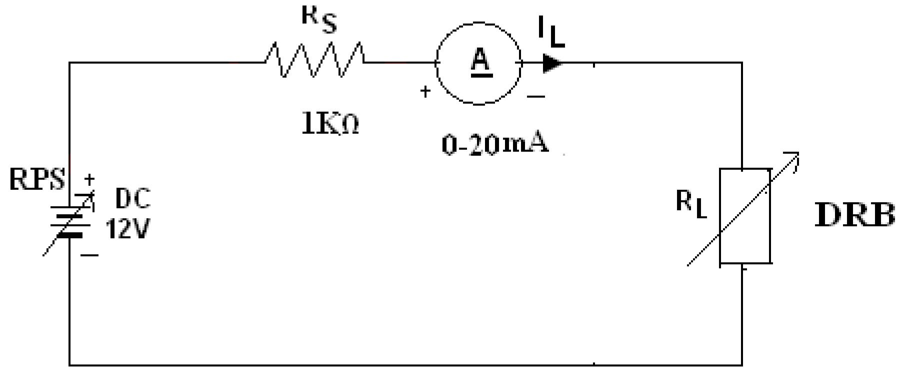
APPARTUS REQUIRED:

SI. No	Equipment	Range	Qty
1	DC Voltage source.	0-30V	1
2	Resistors	470 Ω	1
4	Decade resistance box	0-10k Ω	1
5	Ammeter	0-20mA	1
6	Voltmeter	0-20V	1
7	Connecting wires	1.0.Sq.mm	As required

CIRCUIT DIAGRAM:



PRACTICAL CIRCUIT:



THEORY:

STATEMENT:

It states that the maximum power is transferred from the source to load when the load resistance is equal to the internal resistance of the source.

(or)

The maximum transformer states that "A load will receive maximum power from a linear bilateral network when its load resistance is exactly equal to the Thevenin's resistance of network, measured looking back into the terminals of network.

Consider a voltage source of V of internal resistance R_i delivering power to a load **Resistance R_L**

Circuit current $= \frac{V}{R_L+R_i}$

Power delivered $P = I^2 R_L$

$= \left|\frac{V}{R_L+R_i}\right|^2 R_L$

for maximum poewer $\frac{d(p)}{dt} = 0$

RL+Ri cannot be zero,

Ri – RL = 0

$$\boxed{R_L == Ri}$$

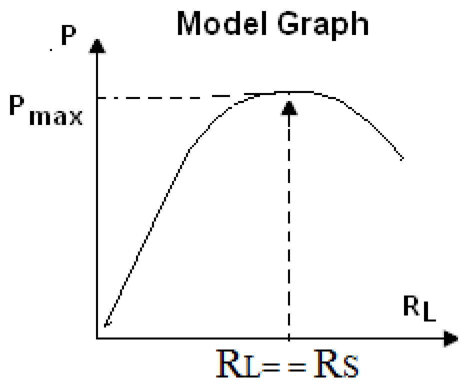
$$P_{max} = \frac{V2}{4RL} \text{ watts}$$

PROCEDURE:

1. Connect the circuit as shown in the above figure.
2. Apply the voltage 12V from RPS.
3. Now vary the load resistance (R_L) in steps and note down the corresponding Ammeter. Reading (I_L)in milli amps and Load Voltage (V_L) volts.
6. Tabulate the readings and find the power for different load resistance values.
7. Draw the graph between Power and Load Resistance.
8. After plotting the graph, the Power will be Maximum, when the Load Resistance will be equal to source Resistance

TABULAR COLUMN:

S.No	R_L(ohms)	I_L(A)	Power(P_L)=I_L^2*R_L(mW)
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			



Theoretical Calculations:-

R = (R_S+ R_L)=………………….**Ω**

I_L = V / R =…………..…….**mA**

Power = (I $_L^2$) **R_L**=…..…..**mW**

RESULT:

QUESTIONS:
1) What is maximum power transfer theorem?
2) What is the application this theorem?

4. VERIFICATION OF THEVENIN'S THEOREM AND NORTON'S THEOREM

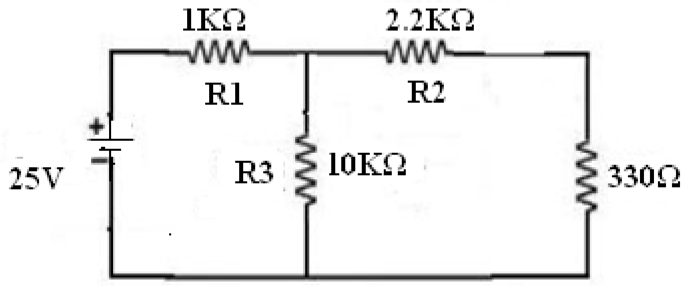
AIM: To verify Theremin's & Norton's theorems for the given circuit.

APPARATUS REQUIRED:

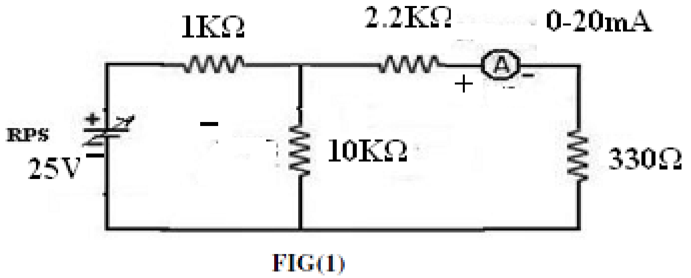
S.No	Name Of The Equipment	Range	Type	Quantity
1	Voltmeter	(0-20)V	Digital	1 NO
2	Ammeter	(0-20)mA	Digital	1 NO
3	RPS	0-30V	Digital	1 NO
4	Resistors	10K Ω,1K Ω		1 NO
		2.2Ω		1 NO
		330 Ω		1 NO
5	Breadboard	-	-	1 NO
6	Connecting wires			Required number

CIRCUIT DIAGRAM:

GIVEN CIRCUIT:

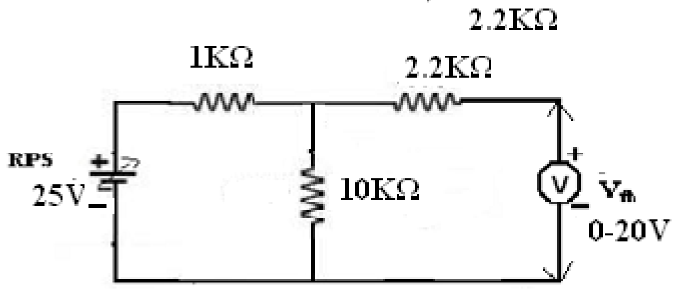


PRACTICAL CIRCUIT DIAGRAMS:
TO FIND I_L:



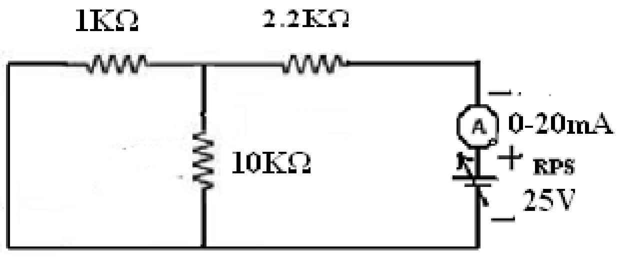
FIG(1)

TO FIND V_{TH}:



FIG(2)

TO FIND R_{th}:



fig(3)

TO FIND I_N:

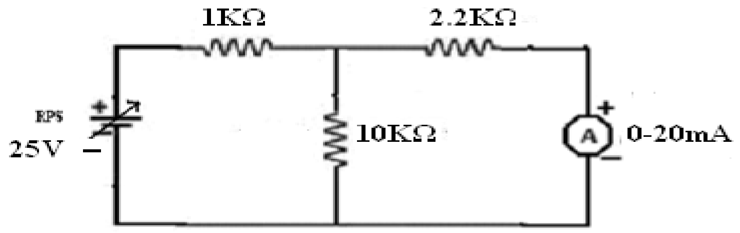


Fig (4)

STATEMENTS:

THEVENIN'S THEOREM:

It states that in any lumped, linear network having more number of sources and elements the equivalent circuit across any branch can be replaced by an equivalent circuit consisting of Theremin's equivalent voltage source Vth in series with Theremin's equivalent resistance Rth. Where Vth is the open circuit voltage across (branch) the two terminals and Rth is the resistance seen from the same two terminals by replacing all other sources with internal resistances.

Thevenin's theorem:

The values of VTh and RTh are determined as mentioned in thevenin's theorem. Once the thevenin equivalent circuit is obtained, then current through any load resistance RL connected across AB is given by, $I = \frac{V_{Th}}{R_{Th}+R_L}$.

Thevenin's theorem is applied to d.c. circuits as stated below.

Any network having terminals A and B can be replaced by a single source of e.m.f. V_{Th} in series with a source resistance R_{Th}

(i) The e.m.f the voltage obtained across the terminals A and B with load, if any removed i.e., it is open circuited voltage between terminals A and B.

(ii) The resistance R_{Th} is the resistance of the network measured between the terminals A and B with load removed and sources of e.m.f replaced by their internal resistances. Ideal voltage sources are replaced with short circuits and ideal current sources are replaced with open circuits.

To find V_{Th}, the load resistor 'RL' is disconnected, then $VTh = \frac{V}{R_1+R_2} X R3$

To find R_{Th},

$$R_{Th} = R2 + \frac{R_1 R_3}{R_1+ R_3}$$

Thevenin's theorem is also called as "Helmoltz theorem"

NORTON'S THEOREM:

Norton's theorem states that in a lumped, linear network the equivalent circuit across any branch is replaced with a current source in parallel a resistance. Where the current is the Norton's current which is the short circuit current though that branch and the resistance is the Norton's resistance which is the equivalent resistance across that branch by replacing all the sources sources with their internal resistances?

Nortom's theorem is applied to d.c circuits may be stated as below.

Any linear network having two terminals 'A' and 'B' can be replaced by a current source of current output IN in parallel with a resistance RN.

(i) The output IN of the current source is equal to the current that would flow through AB when A&B are short circuited.

(ii) The resistance RN is the resistance of network measured b/wn A and B with load removed and the sources of e.m.f replaced by their internal resistances.

Ideal voltage source are replaced with short circuits and ideal current sources are replaced with open circuits .

From the fig:

for source current,

$$I = \frac{V}{R^1} = \frac{V(R_2+R_3)}{R_1R_2+R_1R_3+R_2R_3}$$

for short-circuit current,

$$I_N = I \times \frac{R_3}{R_2+R_3} = \frac{VR_3}{R_1R_2+R_1R_3+R_2R_3}$$

PROCEDURE:

1. Connect the circuit as per fig (1)
2. Adjust the output voltage of the regulated power supply to an appropriate value (Say 25V).
3. Note down the response (current, IL) through the branch of interest i.e. AB (ammeter reading).
4. Reduce the output voltage of the regulated power supply to 0V and switch-off the supply.
5. Disconnect the circuit and connect as per the fig (2).
6. Adjust the output voltage of the regulated power supply to 25V.
7. Note down the voltage across the load terminals AB (Voltmeter reading) that gives Vth.
8. Reduce the output voltage of the regulated power supply to 0V and switch-off the supply.
9. Disconnect the circuit and connect as per the fig (3).
10. Adjust the output voltage of the regulated power supply to an appropriate value (Say V =25V).

11. Note down the current (I) supplied by the source (ammeter reading).
12. The ratio of V and I gives the Rth.
13. Reduce the output voltage of the regulated power supply to 0V and switch-off the supply.
14. Disconnect the circuit and connect as per the fig (4).
15. Adjust the output voltage of the regulated power supply to 25V
16. Note down the response (current, I_N) through the branch AB (ammeter reading).
17. Reduce the output voltage of the regulated power supply to 0V and switch-off the supply.
18. Disconnect the circuit

THERITICAL VALUES:
Tabulation for Thevinen's theorem:

THEORITICAL VALUES	PRACTICAL VALUES
V_{Th}= R_{TH}= I_L=	V_{Th}= R_{TH}= I_L=

Tabulation for Norton's theorem:

THEORITICAL VALUES	PRACTICAL VALUES
I_N= R_N= I_L=	I_N= R_N= I_L=

RESULT:

QUESTIONS:

1) The internal resistance of a source is 2 Ohms and is connected with an External load of 10 Ohms resistance. What is Rth ?
2) In the above question if the voltage is 10 volts and the load is of 50 ohms What is the load current and Vth? Verify IL?
3) If the internal resistance of a source is 5 ohms and is connected with an External load of 25 Ohms resistance. What is Rth?

5. OC & SC TESTS ON 1 – PHASE TRANSFORMER

AIM: To conduct Open circuit and Short circuit tests on 1-phase transformer to pre-determine the efficiency, regulation and equivalent parameters.

NAME PLATE DETAILS:

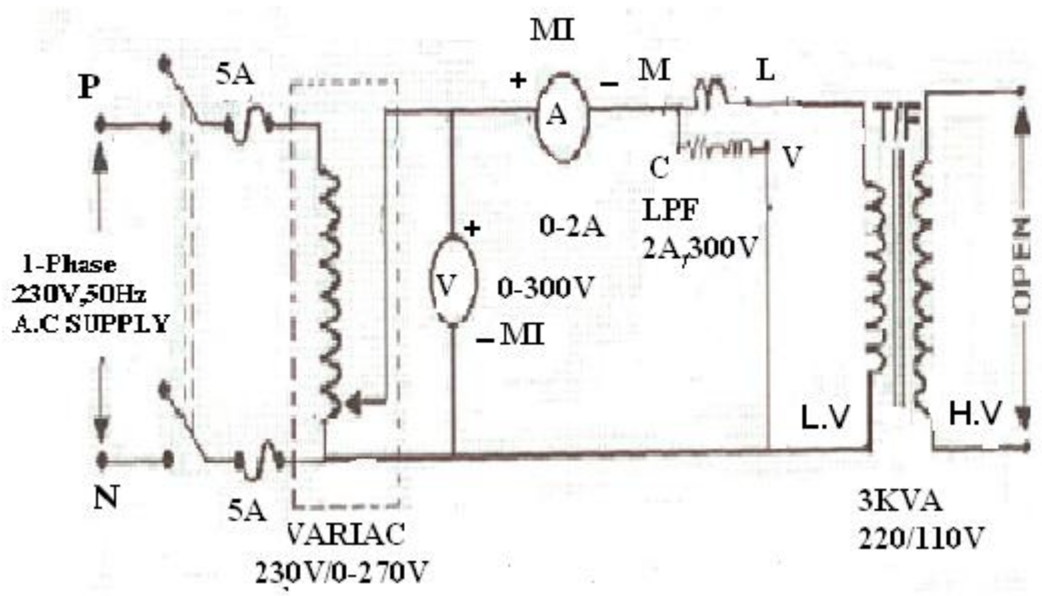
Voltage Ratio	220/110V
Full load Current	13.6A
KVA RATING	3KVA

APPARATUS:

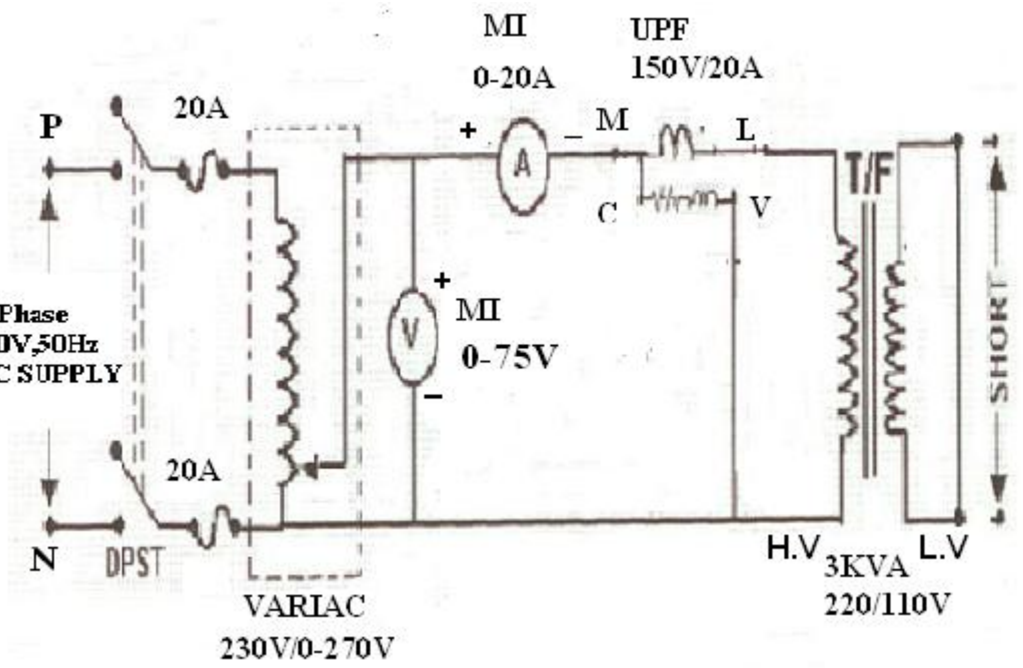
S.NO	Description	Type	Range	Quantity
1	Ammeter	MI	0-20A 0-5A	2no
2	Voltmeter	MI	0-150V 0-300V	2no
3	Wattmeter	LPF UPF	2A,!50V 20A,300V	2no
4	Auto transformer	-	230/0-270V	1no

CIRCUIT DIAGRAM:

OPEN CIRCUIT TEST:



SHORT CIRCUIT TEST:



THEORY:
Transformer is a device which transforms the energy from one circuit to other circuit without change of frequency.
The performance of any transformer calculated by conducting tests .OC and SC tests are conducted on transformer to find the efficiency and regulation of the transformer at any desired power factor.
OC TEST:
The objectives of OC test are
1. To find out the constant losses or iron losses of the transformer.
2. To find out the no load equivalent parameters.
SC TEST:
The objectives of OC test are
1. To find out the variable losses or copper losses of the transformer.
2. To find out the short circuit equivalent parameters.
By calculating the losses and equivalent parameters from the above tests the efficiency and regulation can be calculated at any desired power factor.

PROCEDURE (OC TEST):

1. Connections are made as per the circuit diagram
2. Initially variac should be kept in its minimum position
3. Close the DPST switch
4. By varying Auto transformer bring the voltage to rated voltage
5. When the voltage in the voltmeter is equal to the rated voltage of HV winding note down all the readings of the meters.
6. After taking all the readings bring the variac to its minimum position
7. Now switch off the supply by opening the DPST switch.

PROCEDURE (SC TEST):

1. Connections are made as per the circuit diagram.
2. Short the LV side and connect the meters on HV side.
3. Before taking the single phase, 230 V, 50 Hz supply the variac should be in minimum position.
4. Now close the DPST switch so that the supply is given to the transformer.
5. By varying the variac when the ammeter shows the rated current
(i.e. 13. 6A) then note down all the readings.
6. Bring the variac to minimum position after taking the readings and switch off the supply.

CALCULATIONS:

(a)Calculation of Equivalent circuit parameters:

Let the transformer be the step down transformer.

(i) Parameters calculation from OC test

$\cos \phi_0 = \frac{W_o}{V_o I_o} =$

$I_w = I_0 \cos \phi_0 =$

$R_0 = \frac{V_1}{I_w} =$

$I_\mu = I_0 \sin \phi_0 =$

$X_0 = \frac{V_1}{I_\mu} =$

$K = \frac{V_2}{V_1} =$

(ii) Parameters calculation from SC test

$R_{01} = \frac{W_{SC}}{I_{sc}^2} =$

$X_{01} = \sqrt{Z_{01}^2 - R_{01}^2} =$

$Z_{01} = \frac{V_{SC}}{I_{SC}} =$

(b) Calculations to find efficiency:

For ½ full load

Cupper losses = W_{sc} x $(1/2)^2$ watts =

where W_{sc} = full – load copper losses

Constant losses = W_0 watts =

Output = ½ KVA x cos ϕ = [cos ϕ may be assumed]

Input = output + Cu. Loss + constant loss =

$$\% \ efficiency = \frac{Output}{Input} x\ 100 =$$

(C)Calculation of Regulation at full load:

$$\% \ Regulation = \frac{I_1 R_{01} \cos\phi \pm I_1 X_{01} \sin\phi}{V_1} x\ 100 =$$

'+' for lagging power factors

'—' for leading power factors

O.C TEST OBSERVATIONS:

S.NO	V_0(VOLTS)	I_0(AMPS)	W_0(watts)

S.C TEST OBSERVATIONS:

S.NO	V_{SC}(VOLTS)	I_{SC}(AMPS)	W_{SC}(watts)

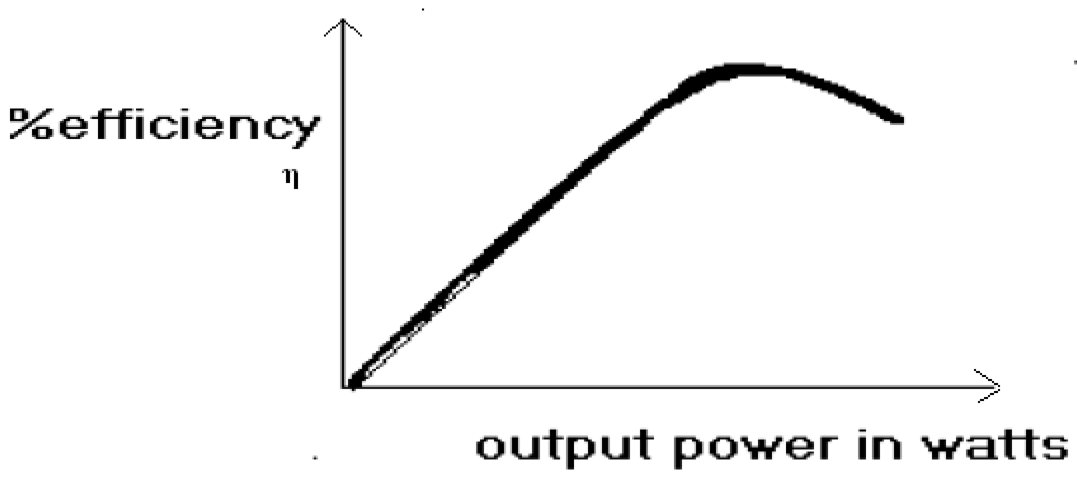
TBULAR COLUMN:

S.NO	% OF LOAD	EFFICIENCY

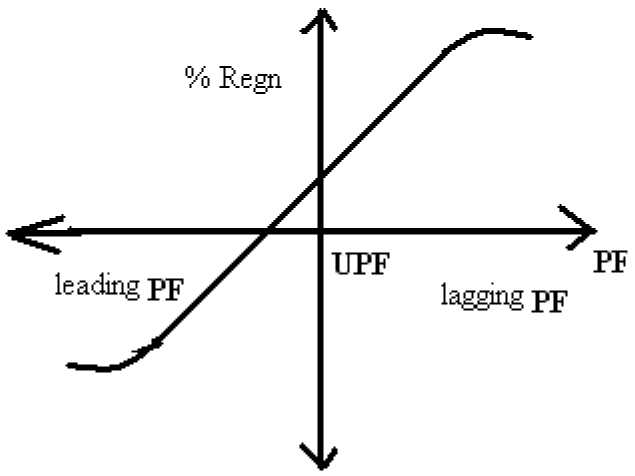
TABULATION:

LAGGING POWER FACTOR			LEADING POWER FACTOR		
SNO	PF	%REG	SNO	PF	%REG
1	0.3			0.3	
2	0.4			0.4	
3	0.5			0.5	
4	0.6			0.6	
5	0.7			0.7	
6	0.8			0.8	
7	0.9			0.9	
8	UNITY			UNITY	

MODEL GRAPHS:
1. EFFICIENCY VS OUTPUT



2. EFFICIENCY VS POWER FACTOR



RESULT:

QUESTIONS:

1) What is a transformer?
2) Draw the equivalent circuit of transformer?
3) What is the efficiency and regulation of transformer?

6. LOAD TEST ON 1-PHASE TRANSFORMER

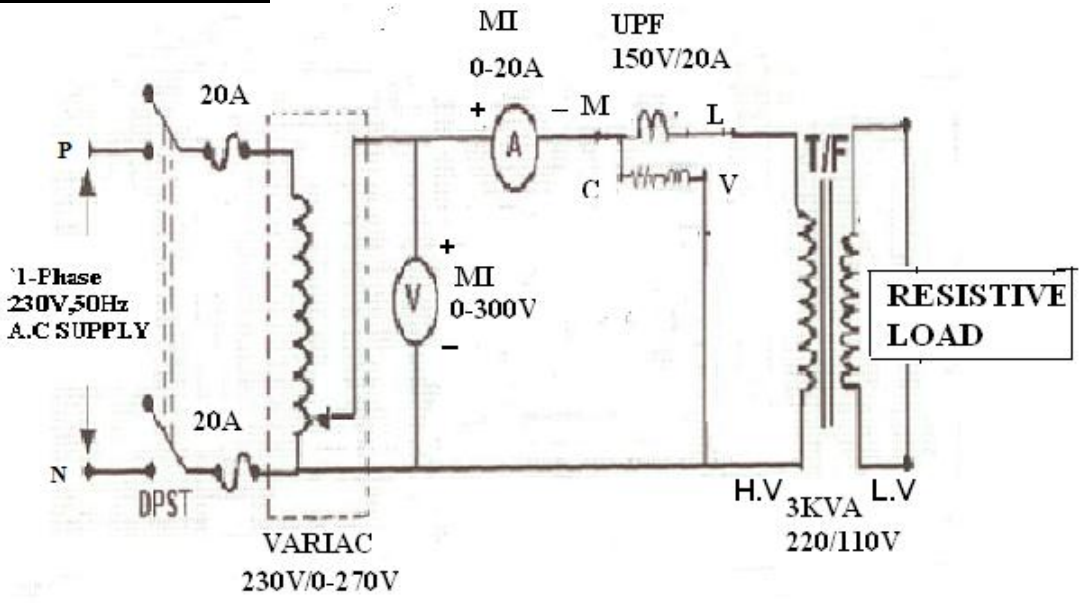
AIM: To find out efficiency by conducting the load test on 1-ϕ Transformer.

APPARATUS:

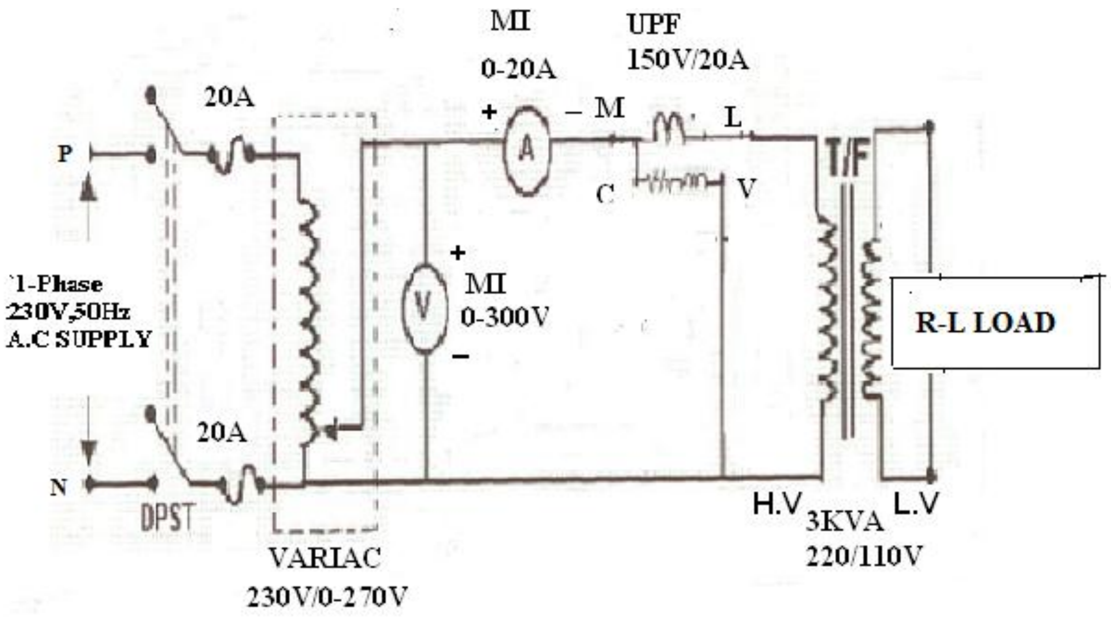
S.NO	APPARATUS	TYPE	RANGE	QUANTITY
1	1-ϕ AUTO Transformer	VARIABLE VOLTAGE	0-270V	01
2	1-ϕ Transformer	Shell type	220/110V	01
3	Voltmeter	MI	0-300V	01
4	Ammeter	MI	0-20A	01
5	Resistive load	Rheostat & variable	0-20A	01
6	Wattmeter	UPF	300V/20A	01
7	Connecting wires			Required number

CIRCUIT DIAGRAM:

RESISTIVE LOAD



R-L LOAD



PROCEDURE:

1) Connect the circuit as shown in above fig.
2) Switch on the input AC supply.
3) Slowly vary the auto transformer knob up to rated input voltage of main transformer.
4) Apply the load slowly up to rated current of the transformer.
5) Take down the voltmeter and ammeter readings.
6) Draw the graph between efficiency and output power.

TABULAR COLUMN (RESISTIVE LOAD):

S.NO	Load Current (amps)	Voltage (volts)

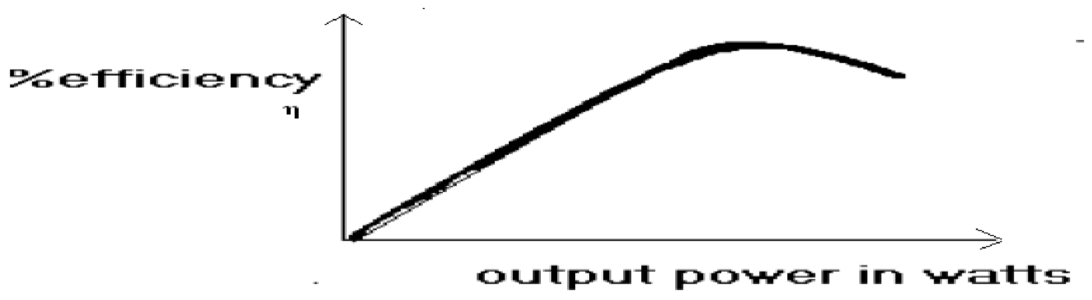
TABULAR COLUMN(R-L LOAD)

S.NO	Load Current (amps)	Voltage (volts)

OBSERVATION TBLE:

S.NO	% OF LOAD	EFFICIENCY

MODEL GRAPHS:
EFFICIENCY VS OUTPUT



RESULT:

QUESTIONS:
1) What is load test on transformer and what is the advantage of this test?
2) What is other test to determine the efficiency and regulation of transformer

PART-B

1. BASIC ELECTRONIC COMPONENTS

1.1. COLOUR CODING OF RESISTOR

Colour Codes are used to identify the value of resistor. The numbers to the Colour are identified in the following sequence which is remembered as **BBROY GREAT BRITAN VERY GOOD WIFE (BBROYGBVGW)** and their assignment is listed in following table.

Black	Brown	Red	Orange	Yellow	Green	Blue	Violet	Grey	White
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Table1: Colour codes of resistor

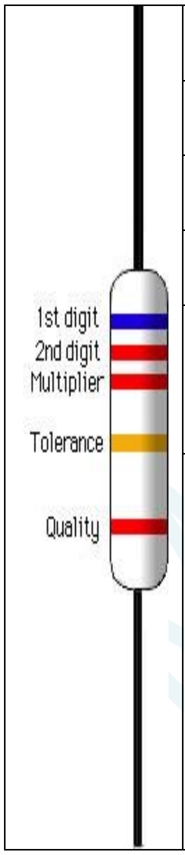
 1st digit 2nd digit Multiplier Tolerance Quality	First find the tolerance band, it will typically be gold (5%) and sometimes silver (10%).
	Starting from the other end, identify the first band - write down the number associated with that color
	Now read the next color, so write down a its vale next to the first value.
	Now read the third or 'multiplier exponent' band and write down that as the number of zeros.
	If the 'multiplier exponent' band is Gold move the decimal point one to the left. If the 'multiplier exponent' band is Silver move the decimal point two places to the left. If the resistor has one more band past the tolerance band it is a quality band.
	Read the number as the '% Failure rate per 1000 hour' This is rated assuming full wattage being applied to the resistors. (To get better failure rates, resistors are typically specified to have twice the needed wattage dissipation that the circuit produces). Some resistors use this band for temco information. 1% resistors have three bands to read digits to the left of the multiplier. They have a different temperature coefficient in order to provide the 1% tolerance. At 1% the temperature coefficient starts to become an important factor. at +/-200 ppm a change in temperature of 25 Deg C causes a value change of up to 1%

Table2: procedure to find the value of resistor using Colour codes

1.2. COLOUR CODING OF CAPACITORS

An electrical device capable of storing electrical energy. In general, a capacitor consists of two metal plates insulated from each other by a dielectric. The capacitance of a capacitor depends primarily upon its shape and size and upon the relative permittivity ε_r of the medium between the plates. In vacuum, in air, and in most gases, ε_r ranges from one to several hundred..

One classification of capacitors comes from the physical state of their dielectrics, which may be gas (or vacuum), liquid, solid, or a combination of these. Each of these classifications may be subdivided according to the specific dielectric used. Capacitors may be further classified by their ability to be used in alternating-current (ac) or direct-current (dc) circuits with various current levels.

- **Capacitor Identification Codes:** There are no international agreements in place to standardize capacitor identification. Most plastic film types (Figure1) have printed values and are normally in microfarads or if the symbol is n, Nanofarads. Working voltage is easily identified. Tolerances are upper case letters: M = 20%, K = 10%, J = 5%, H = 2.5% and F = ± 1pF.

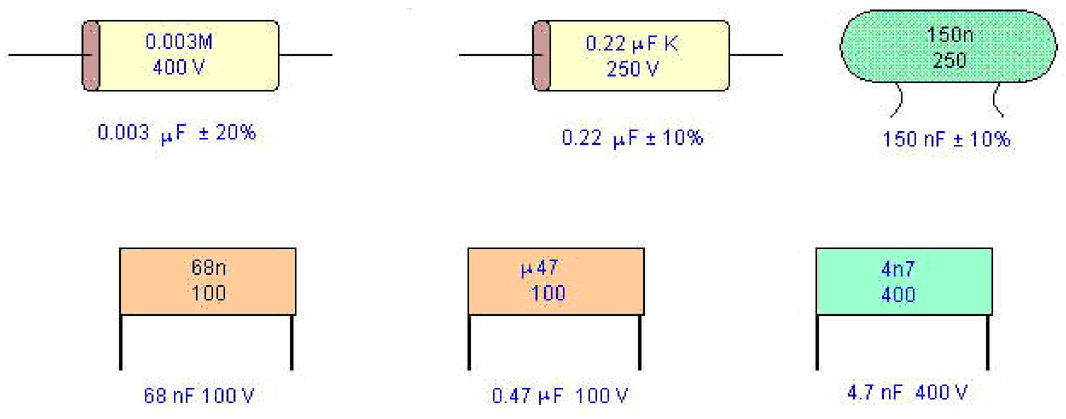


Figure 1: Plastic Film Types

A more difficult scheme is shown in Figure 2 where K is used for indicating Picofarads. The unit is picofarads and the third number is a multiplier. A capacitor coded 474K63 means 47 × 10000 pF which is equivalent to 470000 pF or 0.47 microfarads. K indicates 10% tolerance. 50, 63 and 100 are working volts.

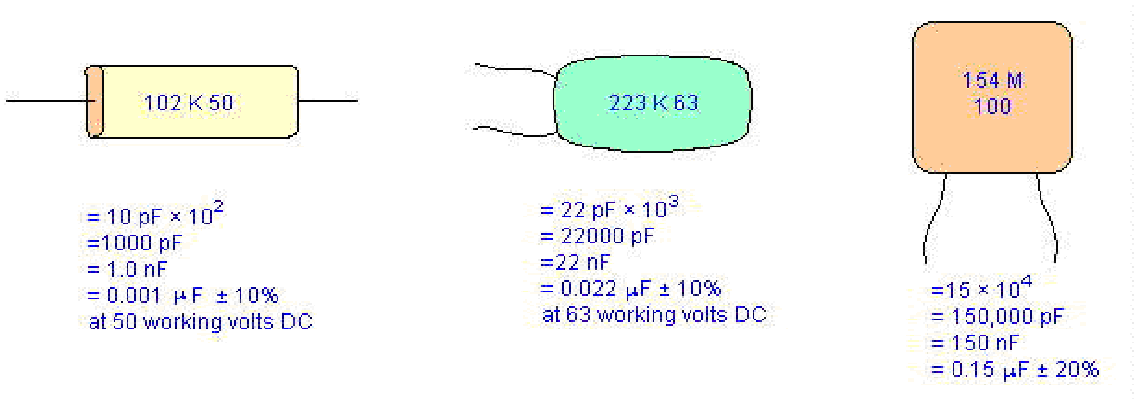


Figure 2: Picofarads Representation

Ceramic disk capacitors have many marking schemes. Capacitance, tolerance, working voltage and temperature coefficient may be found. which is as shown in figure 3. Capacitance values are given as number without any identification as to units. (uF, nF, pF) Whole numbers usually indicate pF and decimal numbers such as 0.1 or 0.47 are microfarads. Odd looking numbers such as 473 is the previously explained system and means 47 nF.

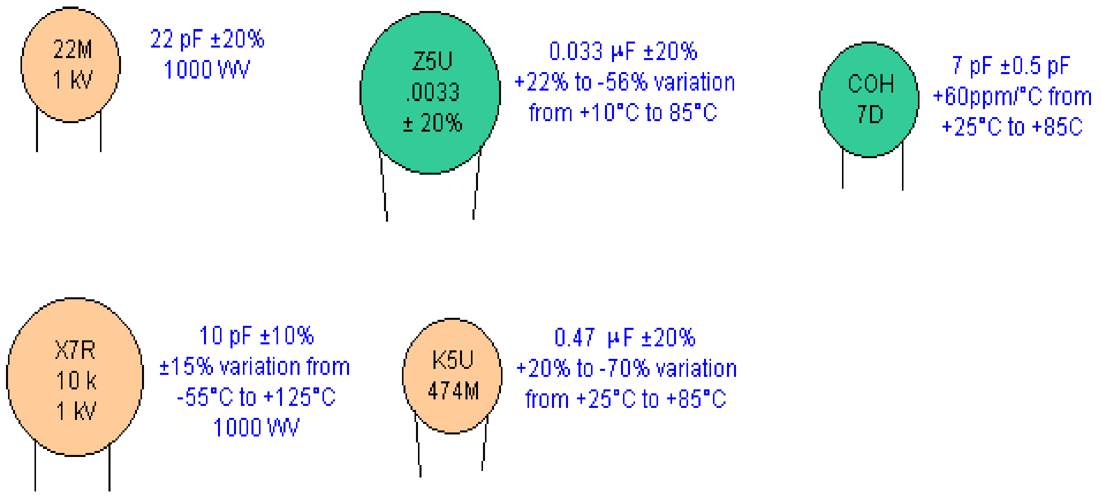


Figure3: ceramic Disk capacitor

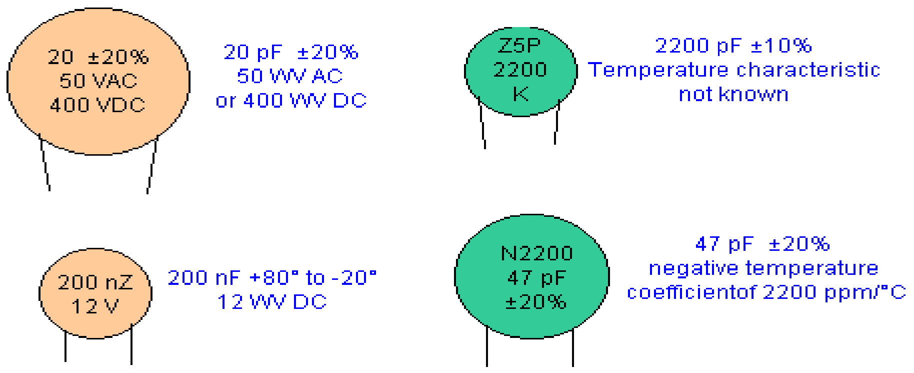


Figure 4: miscellaneous schemes.

- **Electrolytic capacitor properties**

There are a number of parameters of importance beyond the basic capacitance and capacitive reactance when using electrolytic capacitors. When designing circuits using electrolytic capacitors it is necessary to take these additional parameters into consideration for some designs, and to be aware of them when using electrolytic capacitors

- **ESR Equivalent series resistance:**

  Electrolytic capacitors are often used in circuits where current levels are relatively high. Also under some circumstances and current sourced from them needs to have low source impedance, for example when the capacitor is being used in a power supply circuit as a reservoir capacitor. Under these conditions it is necessary to consult the manufacturers' datasheets to discover whether the electrolytic capacitor chosen will meet the requirements for the circuit. If the ESR is high, then it will not be able to deliver the required amount of current in the circuit, without a voltage drop resulting from the ESR which will be seen as a source resistance.

- **Frequency response:**

  One of the problems with electrolytic capacitors is that they have a limited frequency response. It is found that their ESR rises with frequency and this generally limits their use to frequencies below about 100 kHz. This is particularly true for large capacitors, and even the smaller electrolytic capacitors should not be relied upon at high frequencies. To gain exact details it is necessary to consult the manufacturer's data for a given part.

- **Leakage:**

  Although electrolytic capacitors have much higher levels of capacitance for a given volume than most other capacitor technologies, they can also have a higher level of leakage. This is not a problem for most applications, such as when they are used in power supplies. However under some circumstances they are not suitable. For example they should not be used around the input circuitry of an operational amplifier. Here even a small amount of leakage can cause problems because of the high input impedance levels of the op-amp. It is also worth noting that the levels of leakage are considerably higher in the reverse direction.

- **Ripple current:**

  When using electrolytic capacitors in high current applications such as the reservoir capacitor of a power supply, it is necessary to consider the ripple current it is likely to experience. Capacitors have a maximum ripple current they can supply. Above this they can become too hot which will reduce their life. In extreme cases it can cause the capacitor to fail. Accordingly it is necessary to calculate the expected ripple current and check that it is within the manufacturer's maximum ratings.

- **Tolerance:**

  Electrolytic capacitors have a very wide tolerance. Typically this may be -50% + 100%. This is not normally a problem in applications such as decoupling or power supply smoothing, etc. However they should not be used in circuits where the exact value is of importance.

- **Polarization:**

  Unlike many other types of capacitor, electrolytic capacitors are polarized and must be connected within a circuit so that they only see a voltage across them in a particular way.

The physical appearance of electrolytic capacitor is as shown in Figure 5.The capacitors themselves are marked so that polarity can easily be seen. In addition to this it is common for the can of the capacitor to be connected to the negative terminal.

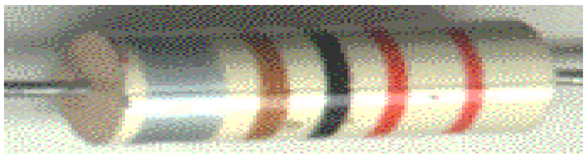
Figure 5: Electrolytic capacitor

It is absolutely necessary to ensure that any electrolytic capacitors are connected within a circuit with the correct polarity. A reverse bias voltage will cause the centre oxide layer forming the dielectric to be destroyed as a result of electrochemical reduction. If this occurs a short circuit will appear and excessive current can cause the capacitor to become very hot. If this occurs the component may leak the electrolyte, but under some circumstances they can explode. As this is not uncommon, it is very wise to take precautions and ensure the capacitor is fitted correctly, especially in applications where high current capability exists.

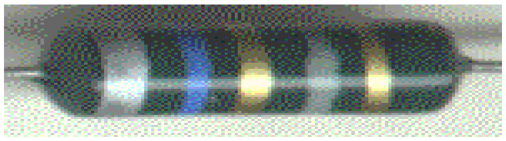
1.3. COLOUR CODING OF INDUCTORS

Inductor is just coil wound which provides more reactance for high frequencies and low reactance for low frequencies.

Molded inductors follow the same scheme except the units are usually micro henries. A brown-black-red inductor is most likely a 1000 uH. Sometimes a silver or gold band is used as a decimal point. So a red-gold-violet inductor would be a 2.7 uH. Also expect to see a wide silver or gold band before the first value band and a thin tolerance band at the end. The typical Colour codes and their values are shown in Figure 6.



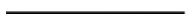
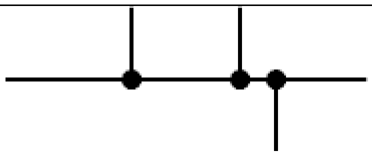
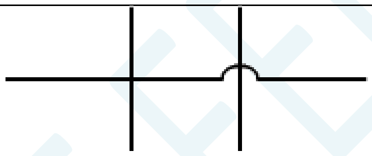
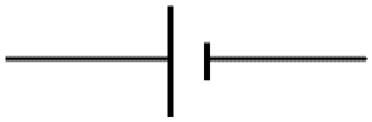
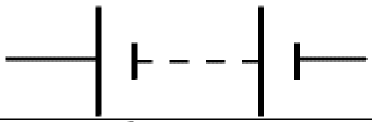
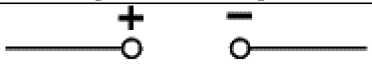
1000uH (1millihenry), 2%

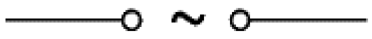
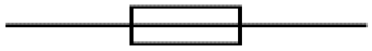
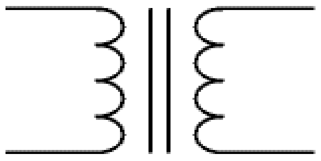
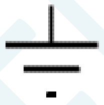
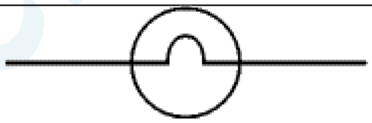
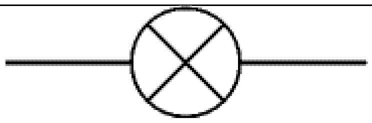
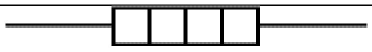


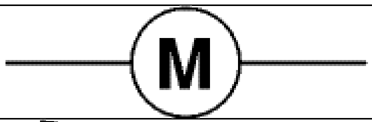
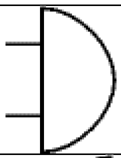
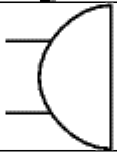
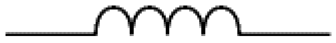
6.8 uH, 5%

Figure 6: Typical inductors Colour coding and their values.

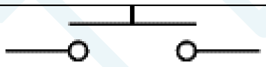
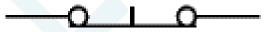
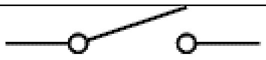
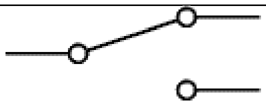
2. CIRCUIT SYMBOLS

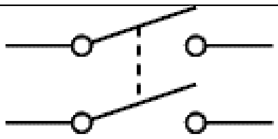
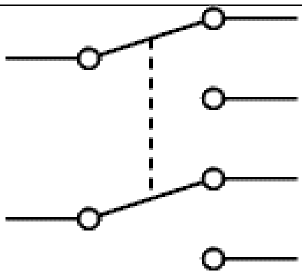
WIRES AND CONNECTIONS			
S.NO.	**COMPONENT NAME**	**CIRCUIT SYMBOL**	**FUNCTION**
1	WIRE		To pass current very easily from one part of a circuit to another.
2	WIRES JOINED		A 'blob' should be drawn where wires are connected (joined), but it is sometimes omitted. Wires connected at 'crossroads' should be staggered slightly to form two T-junctions, as shown on the right.
3	WIRES NOT JOINED		In complex diagrams it is often necessary to draw wires crossing even though they are not connected. I prefer the 'bridge' symbol shown on the right because the simple crossing on the left may be misread as a join where you have forgotten to add a 'blob'.
POWER SUPPLIES			
S.NO	**COMPONENT NAME**	**CIRCUIT SYMBOL**	**FUNCTION**
1.	CELL		Supplies electrical energy. The larger terminal (on the left) is positive (+). A single cell is often called a battery, but strictly a battery is two or more cells joined together
2.	BATTERY		Supplies electrical energy. A battery is more than one cell. The larger terminal (on the left) is positive (+).
3.	DC SUPPLY		Supplies electrical energy. DC = Direct Current, always

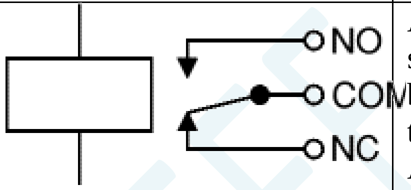
			flowing in one direction.
4.	AC SUPPLY		Supplies electrical energy. AC = Alternating Current, continually changing direction.
5.	FUSE		A safety device which will 'blow' (melt) if the current flowing through it exceeds a specified value.
6.	TRANSFORMER		Two coils of wire linked by an iron core. Transformers are used to step up (increase) and step down (decrease) AC voltages. Energy is transferred between the coils by the magnetic field in the core. There is no electrical connection between the coils.
7.	EARTH(GROUND)		A connection to earth. For many electronic circuits this is the 0V (zero volts) of the power supply, but for mains electricity and some radio circuits it really means the earth. It is also known as ground.
Output Devices: Lamps, Heater, Motor, etc.			
S.NO	**COMPONENT NAME**	**CIRCUIT SYMBOL**	**FUNCTION**
1.	LAMP(LIGHTING)		A transducer which converts electrical energy to light. This symbol is used for a lamp providing illumination, for example a car headlamp or torch bulb
2.	LAMP(INDICATOR)		A transducer which converts electrical energy to light. This symbol is used for a lamp which is an indicator, for example a warning light on a car dashboard.
3.	HEATER		A transducer which converts electrical energy to heat.

4.	MOTOR		A transducer which converts electrical energy to kinetic energy (motion).
5.	BELL		A transducer which converts electrical energy to sound.
6.	BUZZER		A transducer which converts electrical energy to sound.
7.	INDUCTOR(SOLINOID,COIL)		A coil of wire which creates a magnetic field when current passes through it. It may have an iron core inside the coil. It can be used as a transducer converting electrical energy to mechanical energy by pulling on something.

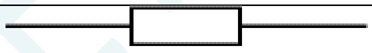
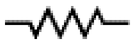
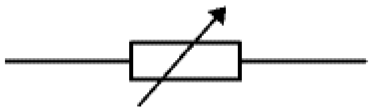
Switches

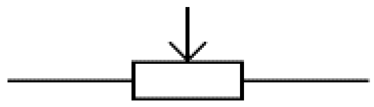
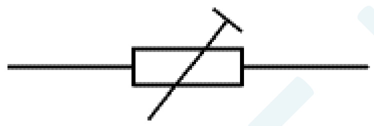
S.NO	COMPONENT NAME	CIRCUIT SYMBOL	FUNCTION
1.	PUSH SWITCH(PUSH TO MAKE)		A push switch allows current to flow only when the button is pressed. This is the switch used to operate a doorbell.
2.	PUSH TO BREAK SWITCH		This type of push switch is normally closed (on), it is open (off) only when the button is pressed.
3.	ON/OFF SWITCH(SPST)		SPST = Single Pole, Single Throw. An on-off switch allows current to flow only when it is in the closed (on) position.
4.	2 WAY SWITCH(SPDT)		SPDT = Single Pole, Double Throw. A 2-way changeover switch directs the flow of current to one of two routes according to its position. Some SPDT switches have a central off position and are described as 'on-off-on'.

5.	DUAL ON-OFF SWITCH(DPST)		DPST = Double Pole, Single Throw. A dual on-off switch which is often used to switch mains electricity because it can isolate both the live and neutral connections.
6.	REVERSING SWITCH(DPDT)		DPDT = Double Pole, Double Throw. This switch can be wired up as a reversing switch for a motor. Some DPDT switches have a central off position.

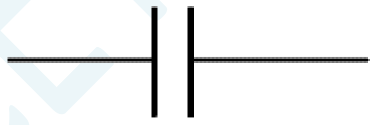
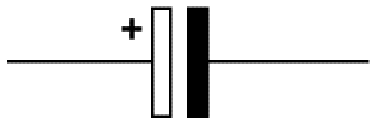
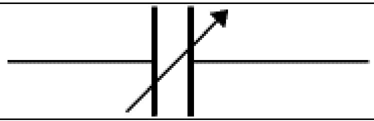
7.	RELAY		An electrically operated switch, for example a 9V battery circuit connected to the coil can switch a 230V AC mains circuit.
			NO = Normally Open,
			COM = Common,
			NC = Normally Closed.

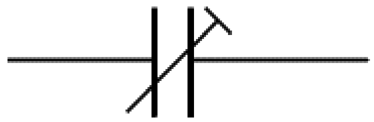
RESISTORS

S.NO	COMPONENT NAME	CIRCUIT SYMBOL	FUNCTION
1.	RESISTOR	 **Or** 	A resistor restricts the flow of current, for example to limit the current passing through an LED. A resistor is used with a capacitor in a timing circuit.
2.	VARIABLE RESISTOR(RHEOSTAT)		This type of variable resistor with 2 contacts (a rheostat) is usually used to control current. Examples include: adjusting lamp brightness, adjusting motor speed, and adjusting the rate of flow of charge into a capacitor in a timing circuit.

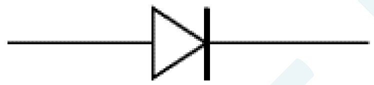
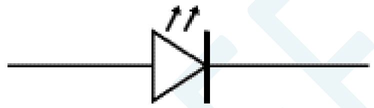
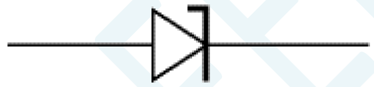
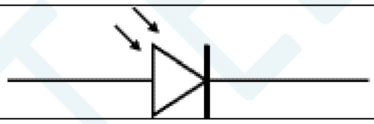
3.	VARIABLE RESISTOR(POTENTIOMETER)		This type of variable resistor with 3 contacts (a potentiometer) is usually used to control voltage. It can be used like this as a transducer converting position (angle of the control spindle) to an electrical signal
4.	VARIABLE RESISTER(PRESET)		This type of variable resistor (a preset) is operated with a small screwdriver or similar tool. It is designed to be set when the circuit is made and then left without further adjustment. Presets are cheaper than normal variable resistors so they are often used in projects to reduce the cost

CAPACITORS

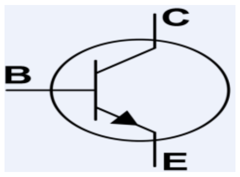
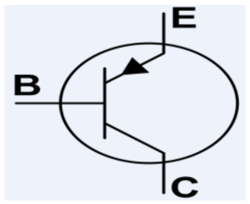
S.NO	NAME OF THE COMPONENT	CIRCUIT SYMBOL	FUNCTION OF THE COMPONENT
1.	CAPACITOR		A capacitor stores electric charge. A capacitor is used with a resistor in a timing circuit. It can also be used as a filter, to block DC signals but pass AC signals.
2.	CAPACITOR POLARISED		A capacitor stores electric charge. This type must be connected the correct way round. A capacitor is used with a resistor in a timing circuit. It can also be used as a filter, to block DC signals but pass AC signals.
3.	VARIABLE CAPACITOR		A variable capacitor is used in a radio tuner.

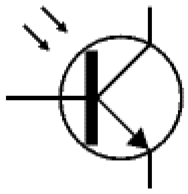
3.	TRIMMER CAPACITOR		This type of variable capacitor (a trimmer) is operated with a small screwdriver or similar tool. It is designed to be set when the circuit is made and then left without further adjustment

DIODES

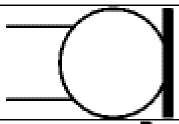
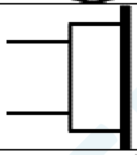
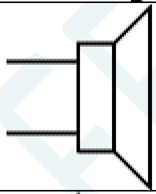
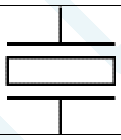
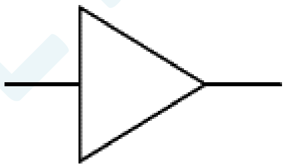
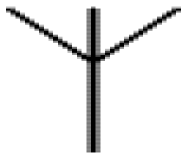
S.NO	NAME OF THE COMPONENT	CIRCUIT SYMBOL	FUNCTION OF THE COMPONENT
1.	DIODE		A device which only allows current to flow in one direction
2.	LED(LIGHT EMITTING DIODE)		A transducer which converts electrical energy to light.
3.	ZENER DIODE		A special diode which is used to maintain a fixed voltage across its terminals
4.	Photodiode		A light-sensitive diode.

TRANSISTORS

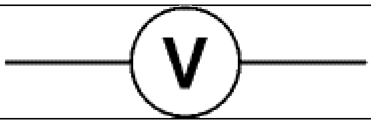
S.NO	NAME OF THE COMPONENT	CIRCUIT SYMBOL	FUNCTION OF THE COMPONENT
5.	TRANSISTOR NPN		A transistor amplifies current. It can be used with other components to make an amplifier or switching circuit.
6.	TRANSISTOR PNP		A transistor amplifies current. It can be used with other components to make an amplifier or switching circuit.

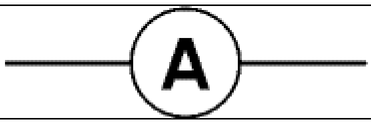
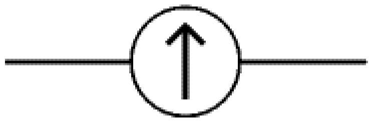
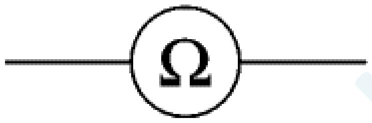
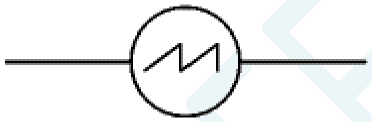
7.	PHOTO TRANSISTOR		A light-sensitive transistor.

AUDIO AND RADIO DEVICES

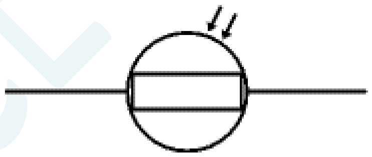
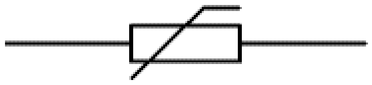
S.NO	NAME OF THE COMPONENT	CIRCUIT SYMBOL	FUNCTION OF THE COMPONENT
1.	MICROPHONE		A transducer which converts sound to electrical energy.
2.	EARPHONE		A transducer which converts electrical energy to sound.
3.	LOUD SPEAKER		A transducer which converts electrical energy to sound.
4.	PIEZO TRANSDUCER		A transducer which converts electrical energy to sound.
5.	AMPLIFIER(GENERAL SYMBOL)		An amplifier circuit with one input. Really it is a block diagram symbol because it represents a circuit rather than just one component.
6.	ARIEL (ANTENNA)		A device which is designed to receive or transmit radio signals. It is also known as an antenna

Meters and Oscilloscope

S.NO	NAME OF THE COMPONENT	CIRCUIT SYMBOL	FUNCTION OF THE COMPONENT
1.	VOLTMETER		A voltmeter is used to measure voltage. The Proper name for voltage is

			'potential difference', but most people prefer to say voltage.
2.	AMMETTER		An ammeter is used to measure current
3.	GALVANOMETER		A galvanometer is a very sensitive meter which is used to measure tiny currents, usually 1mA or less
4.	OHEMMETER		An ohmmeter is used to measure resistance. Most multimeters have an ohmmeter setting.
5.	OSCILLOSCOPE		An oscilloscope is used to display the shape of electrical signals and it can be used to measure their voltage and time period.

Sensors (input devices)

S.NO	NAME OF THE COMPONENT	CIRCUIT SYMBOL	FUNCTION OF THE COMPONENT
1.	LDR		A transducer which converts brightness (light) to resistance (an electrical property). LDR = Light Dependent Resistor
2.	THERMISTOR		A transducer which converts temperature (heat) to resistance (an electrical property).

3. STUDY OF CRO

An oscilloscope is a test instrument which allows us to look at the 'shape' of electrical signals by displaying a graph of voltage against time on its screen. It is like a voltmeter with the valuable extra function of showing how the voltage varies with time. A graticule with a 1cm grid enables us to take measurements of voltage and time from the screen.

The graph, usually called the trace, is drawn by a beam of electrons striking the phosphor coating of the screen making it emit light, usually green or blue. This is similar to the way a television picture is produced.

Oscilloscopes contain a vacuum tube with a cathode (negative electrode) at one end to emit electrons and an anode (positive electrode) to accelerate them so they move rapidly down the tube to the screen. This arrangement is called an electron gun. The tube also contains electrodes to deflect the electron beam up/down and left/right.

The electrons are called cathode rays because they are emitted by the cathode and this gives the oscilloscope its full name of cathode ray oscilloscope or CRO. A dual trace oscilloscope can display two traces on the screen, allowing us to easily compare the input and output of an amplifier for example. It is well worth paying the modest extra cost to have this facility.

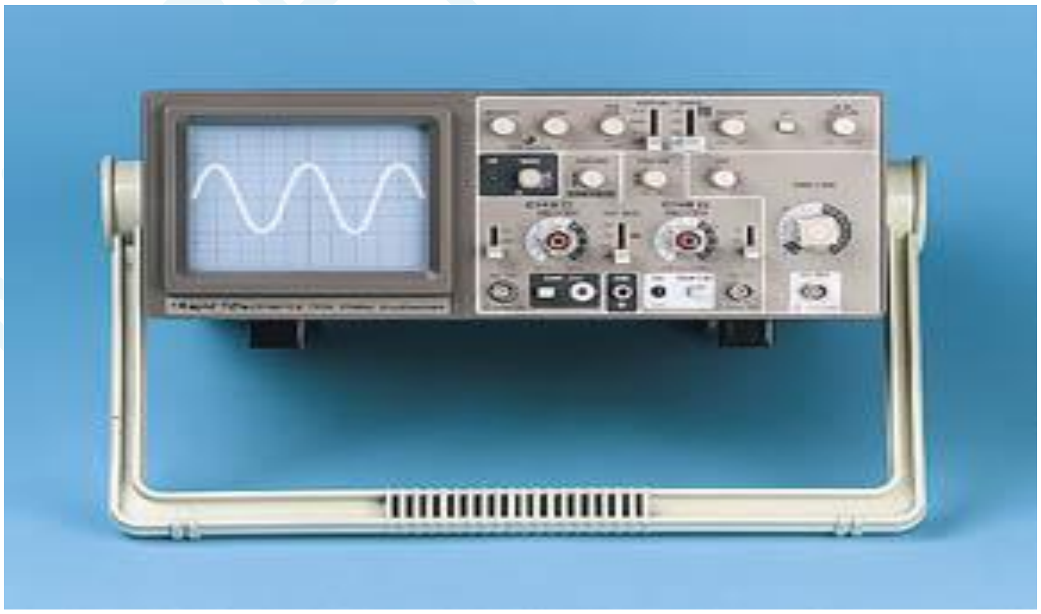


Figure1 : Front Panel of CRO

▪ BASIC OPERATION:

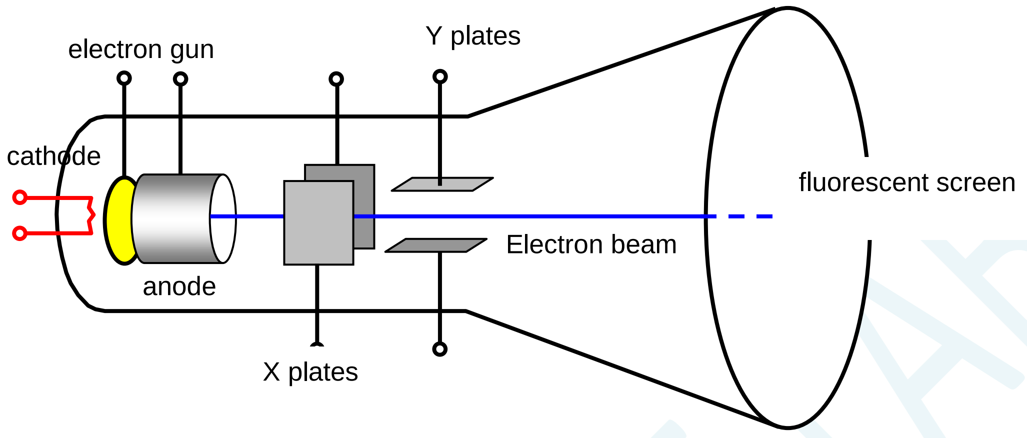


Figure2: Internal Blocks of CRO

➢ Setting up an oscilloscope:

Oscilloscopes are complex instruments with many controls and they require some care to set up and use successfully. It is quite easy to 'lose' the trace off the screen if controls are set wrongly.

There is some variation in the arrangement and labeling of the many controls so the following instructions may need to be adapted for this instrument.

1. Switch on the oscilloscope to warm up (it takes a minute or two).
2. Do not connect the input lead at this stage.
3. Set the AC/GND/DC switch (by the Y INPUT) to DC.
4. Set the SWP/X-Y switch to SWP (sweep).
5. Set Trigger Level to AUTO.
6. Set Trigger Source to INT (internal, the y input).
7. Set the Y AMPLIFIER to 5V/cm (a moderate value).
8. Set the TIMEBASE to 10ms/cm (a moderate speed).
9. Turn the time base VARIABLE control to 1 or CAL.
10. Adjust Y SHIFT (up/down) and X SHIFT (left/right) to give a trace across the middle of the screen, like the picture.
11. Adjust INTENSITY (brightness) and FOCUS to give a bright, sharp trace.

The following type of trace is observed on CRO after setting up, when there is no input signal connected.

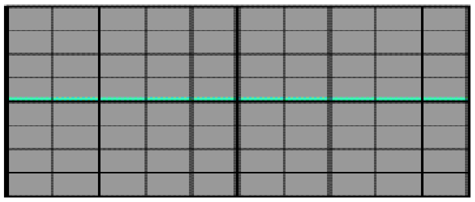


Figure 3: Absence of input signal

- **Connecting an oscilloscope:**

The Y INPUT lead to an oscilloscope should be a co-axial lead and the figure 4 shows its construction. The central wire carries the signal and the screen is connected to earth (0V) to shield the signal from electrical interference (usually called noise).

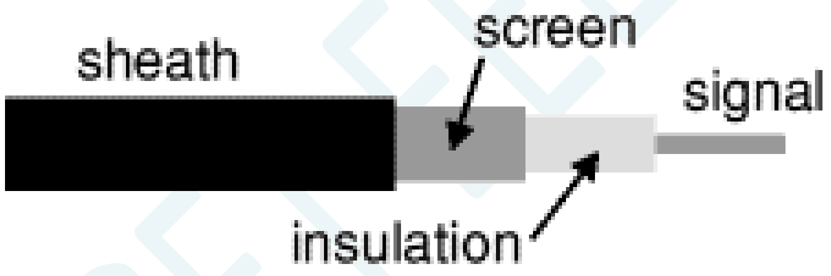


Figure4: Construction of a co-axial lead

Most oscilloscopes have a BNC socket for the y input and the lead is connected with a push and twist action, to disconnect we need to twist and pull. Professionals use a specially designed lead and probes kit for best results with high frequency signals and when testing high resistance circuits, but this is not essential for simpler work at audio frequencies (up to 20kHz).

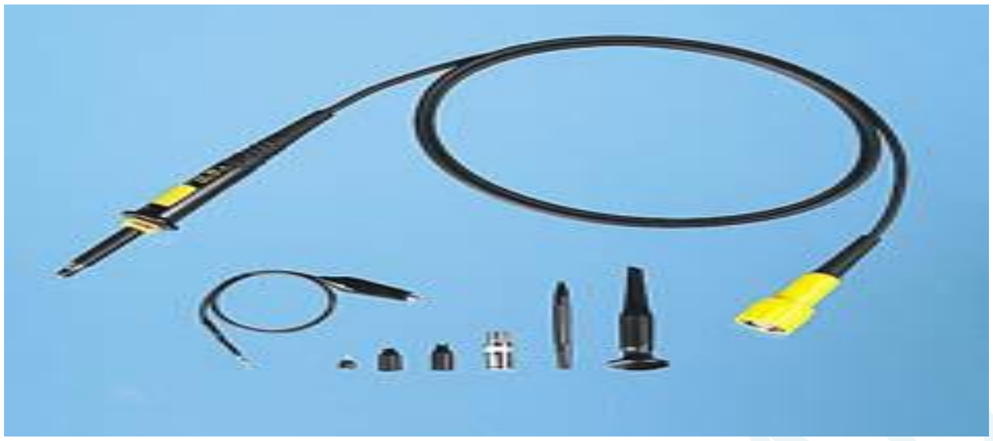


Figure 5: Oscilloscope lead and probes kit

Obtaining a clear and stable trace:

Once if we connect the oscilloscope to the circuit, it is necessary to adjust the controls to obtain a clear and stable trace on the screen in order to test it.

- The Y AMPLIFIER (VOLTS/CM) control determines the height of the trace. Choose a setting so the trace occupies at least half the screen height, but does not disappear off the screen.
- The TIMEBASE (TIME/CM) control determines the rate at which the dot sweeps across the screen. Choose a setting so the trace shows at least one cycle of the signal across the screen. Note that a steady DC input signal gives a horizontal line trace for which the time base setting is not critical.
- The TRIGGER control is usually best left set to AUTO.

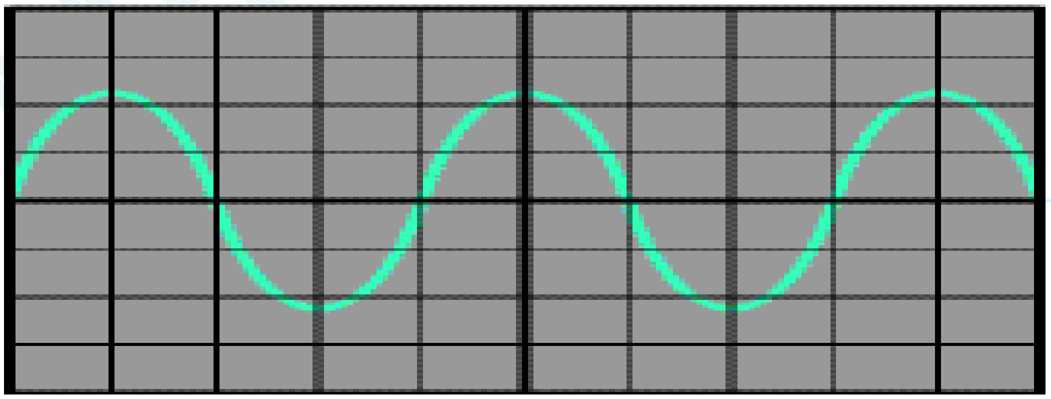


Figure 6 : Stable waveform

➢ Measuring voltage and time period

The trace on an oscilloscope screen is a graph of voltage against time. The shape of this graph is determined by the nature of the input signal. In addition to the properties labeled on the graph, there is frequency which is the number of cycles per second. The diagram shows a sine wave but these properties apply to any signal with a constant shape

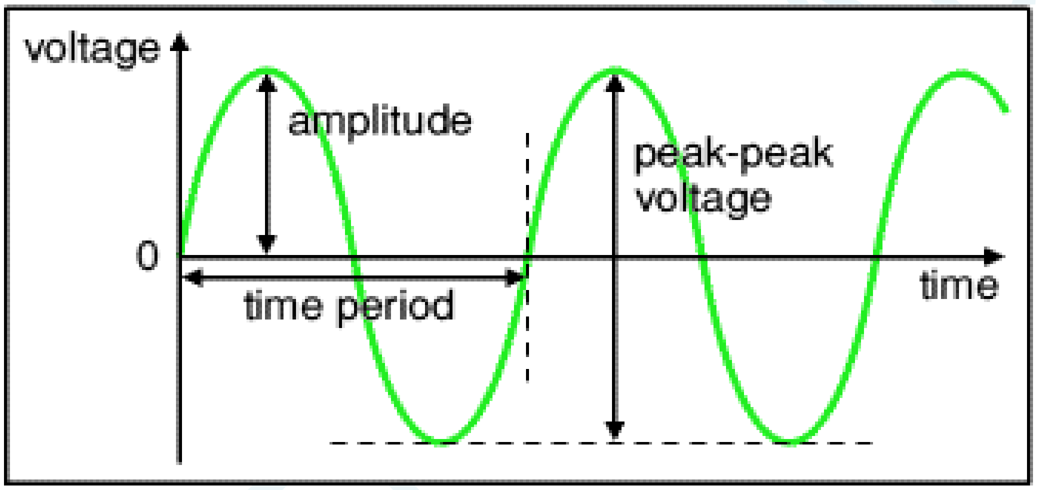


Figure7: Properties of trace

- **Amplitude** is the maximum voltage reached by the signal. It is measured in volts.
- **Peak voltage** is another name for amplitude.
- **Peak-peak voltage** is twice the peak voltage (amplitude). When reading an oscilloscope trace it is usual to measure peak-peak voltage.
- **Time period** is the time taken for the signal to complete one cycle.
  It is measured in seconds (s), but time periods tend to be short so milliseconds (ms) and microseconds (µs) are often used. 1ms = 0.001s and
  1µs = 0.000001s.
- **Frequency** is the number of cycles per second. It is measured in hertz (Hz), but frequencies tend to be high so kilohertz (kHz) and megahertz (MHz) are often used. 1kHz = 1000Hz and 1MHz = 1000000Hz.

$$\text{Frequency} = \frac{1}{\text{Time period}}$$

$$\text{Time period} = \frac{1}{\text{Frequency}}$$

A) Voltage: Voltage is shown on the vertical y-axis and the scale is determined by the Y AMPLIFIER (VOLTS/CM) control. Usually peak-peak voltage is measured because it can be read correctly even if the position of 0V is not known. The amplitude is half the peak-peak voltage.

Voltage = distance in cm × volts/cm

B) Time period: Time is shown on the horizontal x-axis and the scale is determined by the TIMEBASE (TIME/CM) control. The time period (often just called period) is the time for one cycle of the signal. The frequency is the number of cycles per second, frequency = 1/time period.

Time = distance in cm × time/cm

4. STUDY OF FUNCTION GENERATOR

A function generator is a device that can produce various patterns of voltage at a variety of frequencies and amplitudes. It is used to test the response of circuits to common input signals. The electrical leads from the device are attached to the ground and signal input terminals of the device under test.

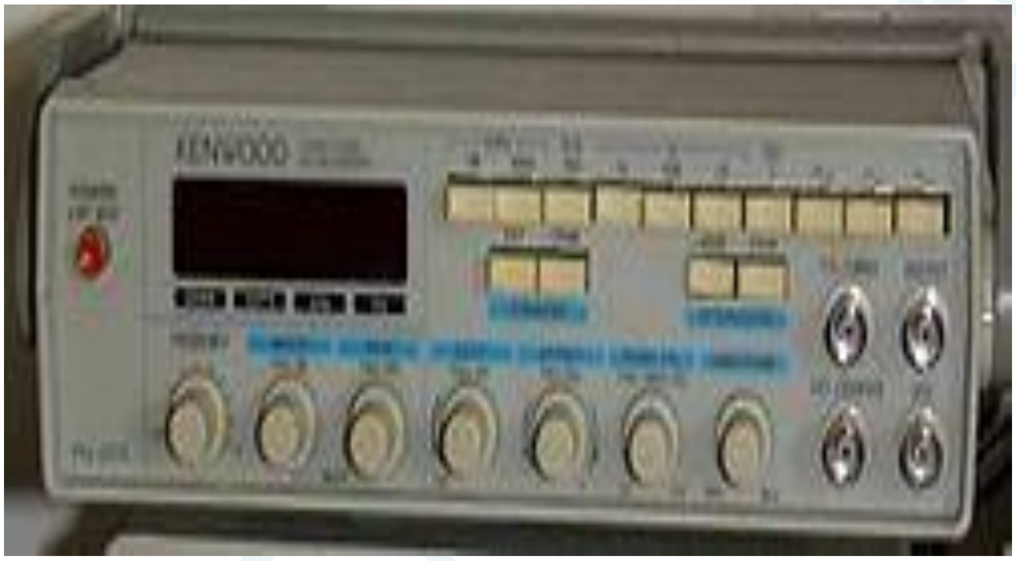


Figure 1: A typical low-cost function generator.

Features and controls :

Most function generators allow the user to choose the shape of the output from a small number of options.

- Square wave - The signal goes directly from high to low voltage.

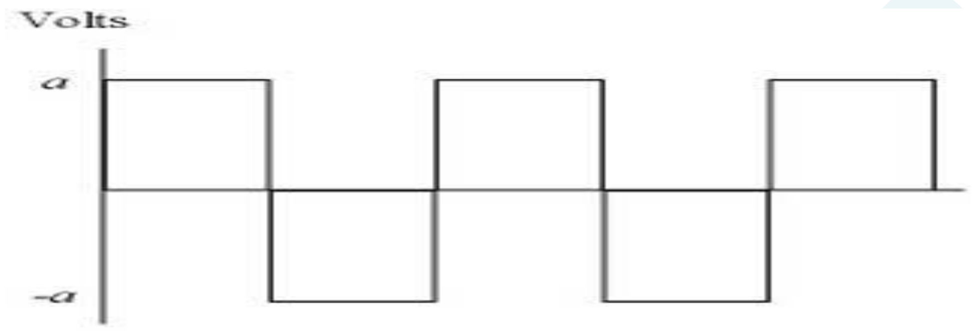


Figure 2: Square wave

The duty cycle of a signal refers to the ratio of high voltage to low voltage time in a square wave signal.

- Sine wave - The signal curves like a sinusoid from high to low voltage.

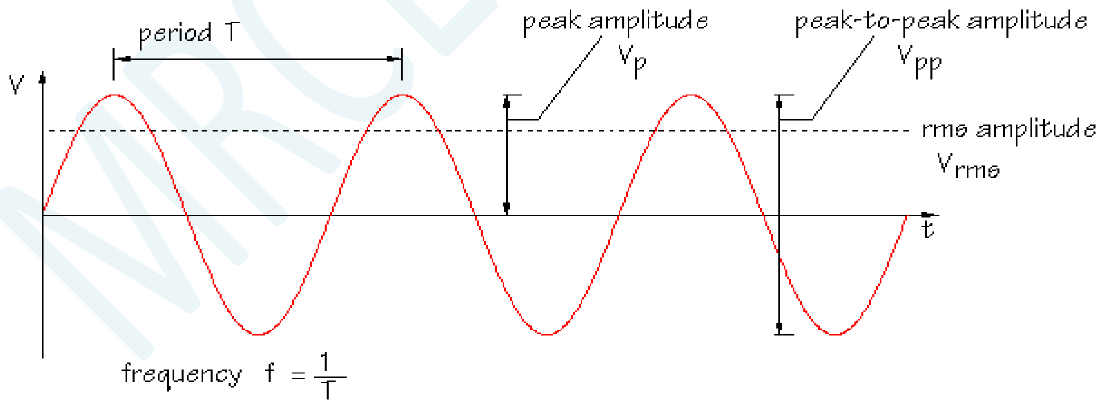


Figure3: Sine Wave

- Triangle wave - The signal goes from high to low voltage at a fixed rate.

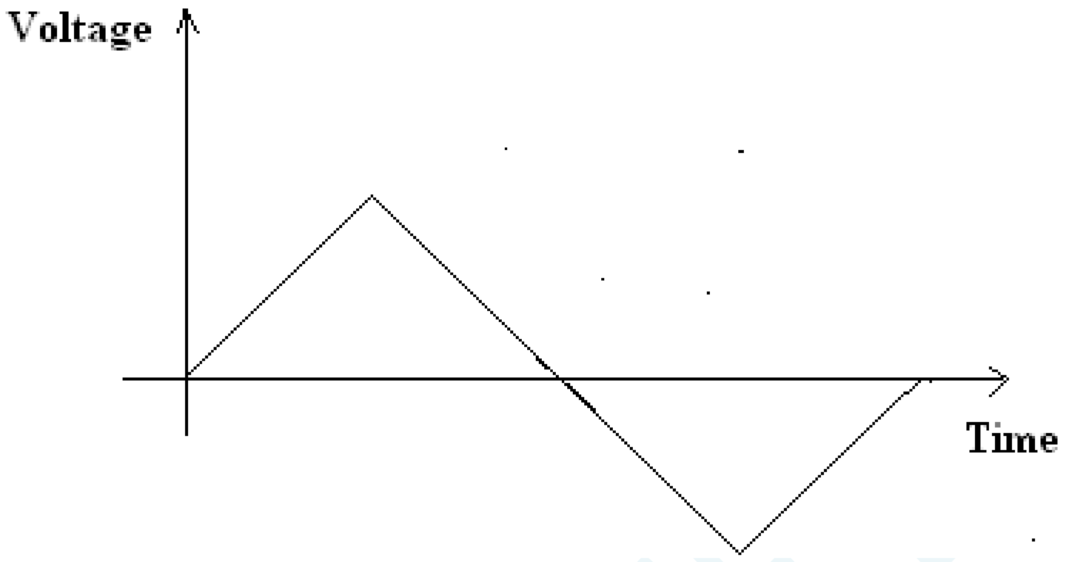


Figure4: Triangular Wave

The amplitude control on a function generator varies the voltage difference between the high and low voltage of the output signal. The direct current (DC) offset control on a function generator varies the average voltage of a signal relative to the ground.

The frequency control of a function generator controls the rate at which output signal oscillates. On some function generators, the frequency control is a combination of different controls. One set of controls chooses the broad frequency range (order of magnitude) and the other selects the precise frequency. This allows the function generator to handle the enormous variation in frequency scale needed for signals.

➢ How to use a function generator

After powering on the function generator, the output signal needs to be configured to the desired shape. Typically, this means connecting the signal and ground leads to an oscilloscope to check the controls. Adjust the function generator until the output signal is correct, then attach the signal and ground leads from the function generator to the input and ground of the device under test. For some applications, the negative lead of the function generator should attach to a negative input of the device, but usually attaching to ground is sufficient.

5. STUDY OF REGULATED POWER SUPPLY

There are many types of power supply. Most are designed to convert high voltage AC mains electricity to a suitable low voltage supply for electronic circuits and other devices. A power supply can by broken down into a series of blocks, each of which performs a particular function. For example a 5V regulated supply:

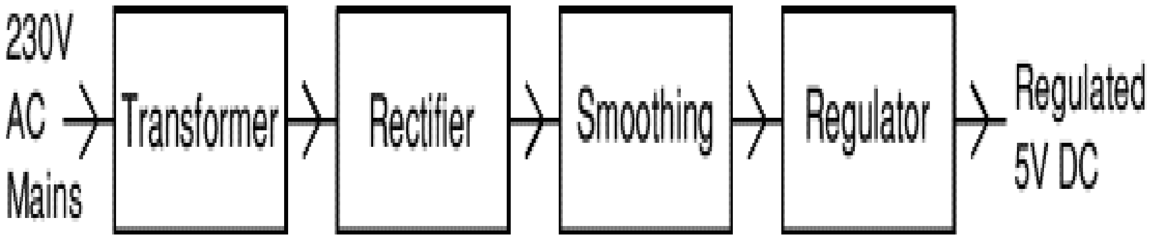


Figure1: Block Diagram of Regulated power supply

Each of the blocks is described in more detail below:

- Transformer: Steps down high voltage AC mains to low voltage AC.
- Rectifier: Converts AC to DC, but the DC output is varying.
- Smoothing: Smooths the DC from varying greatly to a small ripple.
- Regulator: Eliminates ripple by setting DC output to a fixed voltage.

➢ Dual Supplies:

Some electronic circuits require a power supply with positive and negative outputs as well as zero volts (0V). This is called a 'dual supply' because it is like two ordinary supplies connected together as shown in the diagram. Dual supplies have three outputs, for example a ±9V supply has +9V, 0V and -9V outputs.

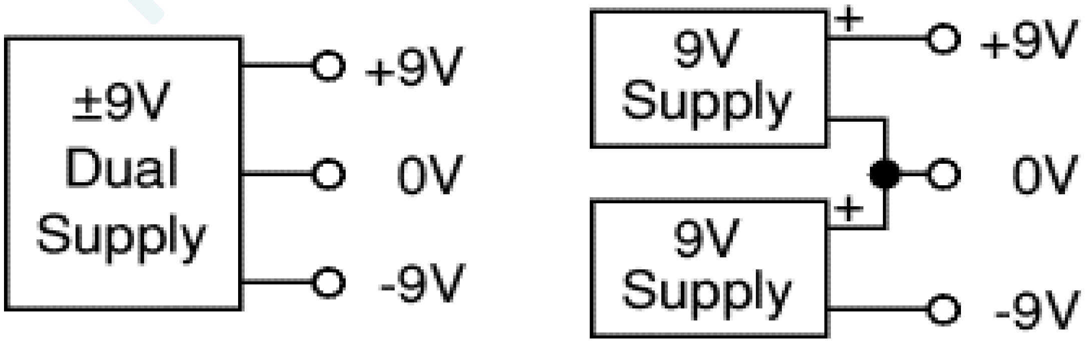


Figure 2: Dual Supply

6. TYPES OF CIRCUIT BOARD

- **Breadboard:**

  This is a way of making a temporary circuit, for testing purposes or to try out an idea. No soldering is required and all the components can be re-used afterwards. It is easy to change connections and replace components. Almost all the Electronics Club projects started life on a breadboard to check that the circuit worked as intended. The following figure depicts the appearance of Bread board in which the holes in top and bottom stribes are connected horizontally that are used for power supply and ground connection conventionally and holes on middle stribes connected vertically. And that are used for circuit connections conventionally.

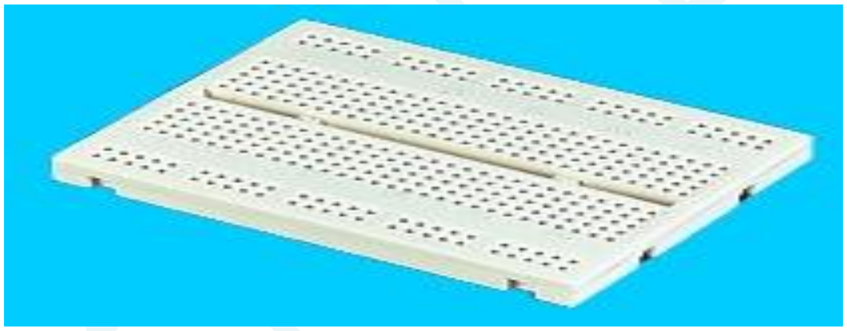


Figure 1: Bread board

- **Strip board:**

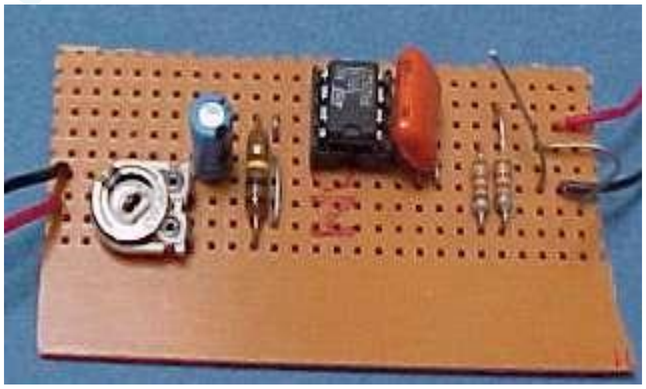


Figure 2: Strib board

Strip board has parallel strips of copper track on one side. The strips are 0.1" (2.54mm) apart and there are holes every 0.1" (2.54mm). Strip board requires no special preparation other than cutting to size. It can be cut with a junior hacksaw, or simply snap it along the lines of holes by putting it over the edge of a bench or table and pushing hard.

Printed Circuit Board: A printed circuit board, or PCB, is used to mechanically support and electrically connect electronic components using conductive pathways, tracks or traces etched from copper sheets laminated onto a non-conductive substrate. It is also referred to as printed wiring board (PWB) or etched wiring board. A PCB populated with electronic components is a printed circuit assembly (PCA), also known as a printed circuit board assembly (PCBA).

Printed circuit boards have copper tracks connecting the holes where the components are placed. They are designed specially for each circuit and make construction very easy. However, producing the PCB requires special equipment so this method is not recommended if you are a beginner unless the PCB is provided for you.

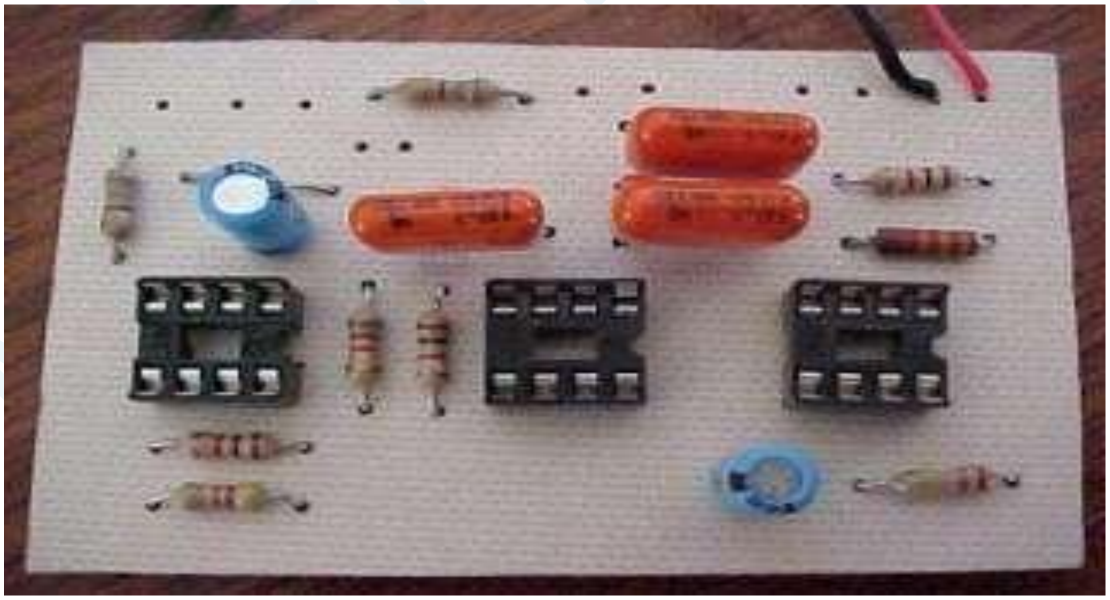


Figure 3: Printed circuit board

PCBs are inexpensive, and can be highly reliable. They require much more layout effort and higher initial cost than either wire-wrapped or point-to-point constructed circuits, but are much cheaper and faster for high-volume production. Much of the electronics industry's PCB design, assembly, and quality control needs are set by standards that are published by the IPC organization.

7. P-N JUNCTION DIODE CHARACTERISTICS

AIM:

1. To observe and draw the Forward and Reverse bias V-I Characteristics of a P-N Junction diode.
2. To calculate static and dynamic resistance in both forward and Reverse Bias conditions.

APPARATUS:

1. P-N Diode IN4007	-	1No.
2. Regulated Power supply (0-30V)	-	1No.
3. Resistor 1KΩ	-	1No.
4. Ammeter (0-20 mA)	-	1No
5. Ammeter (0-200µA)	-	1No.
6. Voltmeter (0-20V)	-	2No.
7. Bread board		
8. Connecting wires		

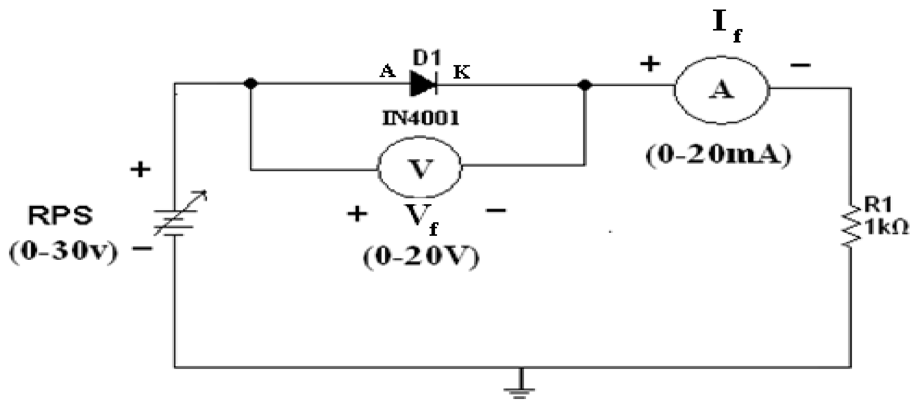
THEORY:

A p-n junction diode conducts only in one direction. The V-I characteristics of the diode are curve between voltage across the diode and current flowing through the diode. When external voltage is zero, circuit is open and the potential barrier does not allow the current to flow. Therefore, the circuit current is zero. When P-type (Anode) is connected to +ve terminal and n- type (cathode) is connected to –ve terminal of the supply voltage is known as forward bias. The potential barrier is reduced when diode is in the forward biased condition. At some forward voltage, the potential barrier altogether eliminated and current starts flowing through the diode and also in the circuit. Then diode is said to be in ON state. The current increases with increasing forward voltage.

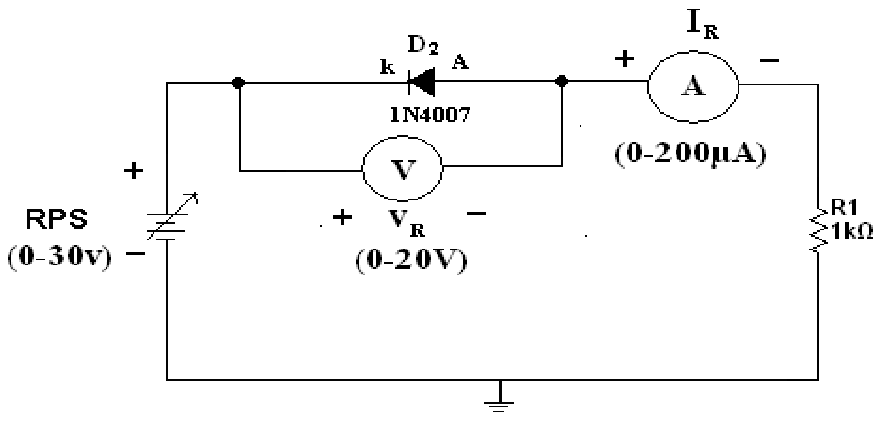
When N-type (cathode) is connected to +ve terminal and P-type (Anode) is connected –ve terminal of the supply voltage is known as reverse bias and the potential barrier across the junction increases. Therefore, the junction resistance becomes very high and a very small current (reverse saturation current) flows in the circuit. Then diode is said to be in OFF state. The reverse bias current is due to minority charge carriers.

CIRCUIT DIAGRAM:

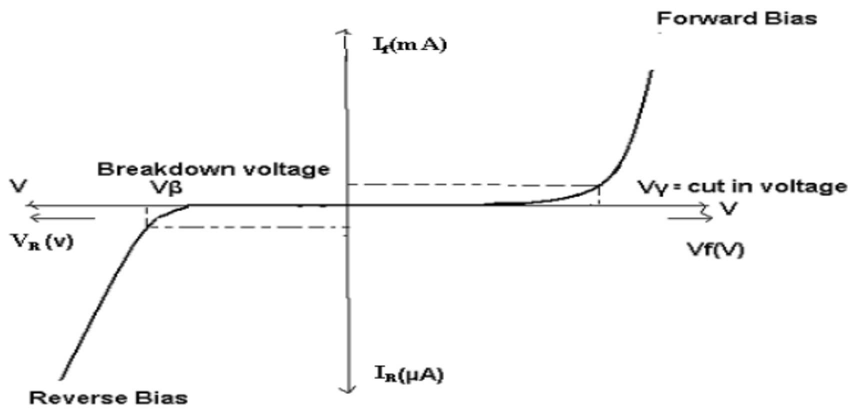
A) Forward bias:



B) Reverse Bias:



MODEL GRAPH:



PROCEDURE:

A) FORWARD BIAS:

1. Connections are made as per the circuit diagram.
2. For forward bias, the RPS +ve is connected to the anode of the diode and RPS –ve is connected to the cathode of the diode
3. Switch on the power supply and increases the input voltage (supply voltage) in Steps of 0.1V
4. Note down the corresponding current flowing through the diode and voltage across the diode for each and every step of the input voltage.
5. The reading of voltage and current are tabulated.
6. Graph is plotted between voltage (V_f) on X-axis and current (I_f) on Y-axis.

B) REVERSE BIAS:

1. Connections are made as per the circuit diagram
2. For reverse bias, the RPS +ve is connected to the cathode of the diode and RPS –ve is connected to the anode of the diode.
3. Switch on the power supply and increase the input voltage (supply voltage) in Steps of 1V.

4. Note down the corresponding current flowing through the diode voltage across the diode for each and every step of the input voltage.
5. The readings of voltage and current are tabulated
6. Graph is plotted between voltage (V_R) on X-axis and current (I_R) on Y-axis.

PRECAUTIONS:

1. All the connections should be correct.
2. Parallax error should be avoided while taking the readings from the Analog meters.

VIVA QUESTIONS:

1. Define depletion region of a diode?
2. What is meant by transition & space charge capacitance of a diode?
3. Is the V-I relationship of a diode Linear or Exponential?
4. Define cut-in voltage of a diode and specify the values for Si and Ge diodes?
5. What are the applications of a p-n diode?
6. Draw the ideal characteristics of P-N junction diode?
7. What is the diode equation?
8. What is PIV?
9. What is the break down voltage?
10. What is the effect of temperature on PN junction diodes?

OBSERVATIONS:

A) FORWARD BIAS:

S.NO	Applied Voltage(V)	Forward Voltage(V_f)	Forward Current(I_f(mA))

B) REVERSE BIAS:

S.NO	Applied Voltage(V)	Reverse Voltage(V_R)	Reverse Current(I_R(µA))

RESULT:

Calculating Static and Dynamic Resistance of given diode.

In forward bias condition:

Static Resistance , $R_s = Vf/I_f =$

Dynamic Resistance, $R_D = \Delta V_f / \Delta I_f =$

In Reverse bias condition:

Static Resistance , $R_s = V_R/I_R =$

Dynamic Resistance, $R_D = \Delta V_R / \Delta I_R =$

8. ZENER DIODE CHARACTERISTICS

AIM:

To observe and draw the static characteristics of a zener diode

APPARATUS:

1. Zener diode	-1No.
2. Regulated Power Supply (0-30v)	-1No.
3. Voltmeter (0-20v)	-1No.
4. Ammeter (0-20mA)	-1No.
5. Resistor (1K ohm)	
6. Bread Board	
7. Connecting wires	

THEORY:

A zener diode is heavily doped p-n junction diode, specially made to operate in the break down region. A p-n junction diode normally does not conduct when reverse biased. But if the reverse bias is increased, at a particular voltage it starts conducting heavily. This voltage is called Break down Voltage. High current through the diode can permanently damage the device

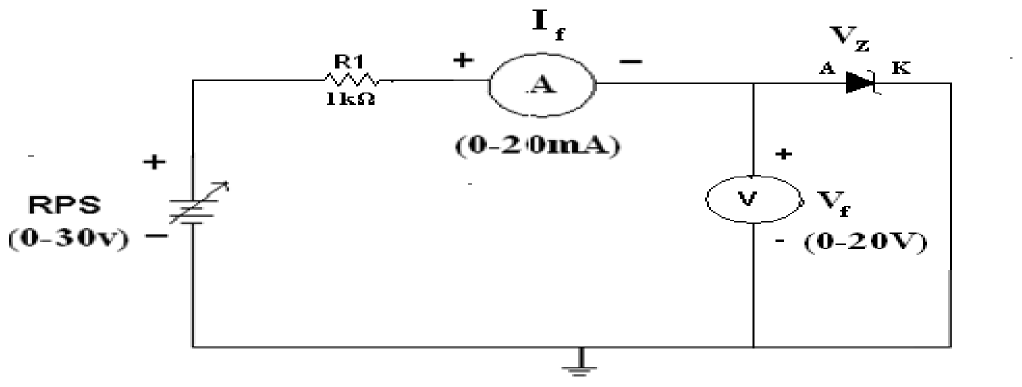
To avoid high current, we connect a resistor in series with zener diode. Once the diode starts conducting it maintains almost constant voltage across the terminals whatever may be the current through it, i.e., it has very low dynamic resistance. It is used in voltage regulators.

PROCEDURE :

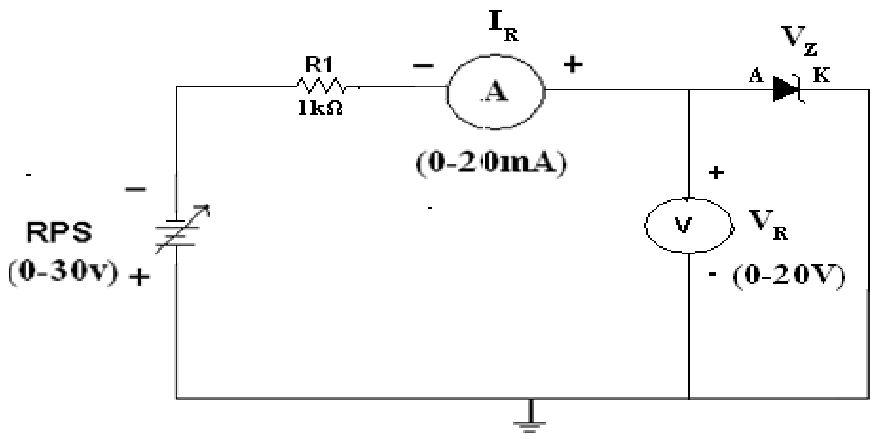
1. Connections are made as per the circuit diagram.
2. The Regulated power supply voltage is increased in steps.
3. The Forward current (l_f), and the forward voltage (V_f.) are observed and then noted in the tabular form.
4. A graph is plotted between Forward current (l_f) on X-axis and the forward voltage (V_f) on Y-axis.

CIRCUIT DIAGRAM :

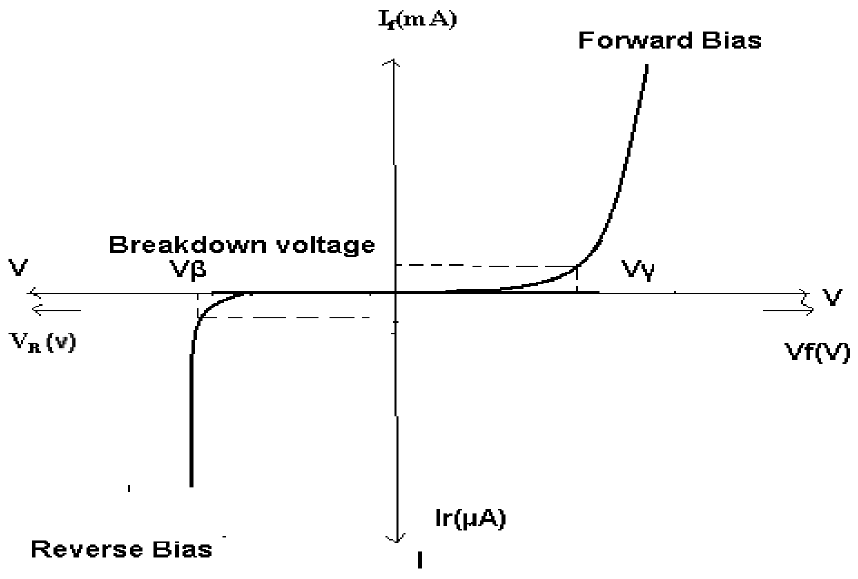
A) FORWARD CHARACTERISTICS :



B) REVERSE CHARACTERISTICS:



Model Graph:



PRECAUTIONS:

1. The terminals of the zener diode should be properly identified
2. While determined the load regulation, load should not be immediately shorted.
3. Should be ensured that the applied voltages & currents do not exceed the ratings of the diode.

VIVAQUESTIONS:

1. What type of temp? Coefficient does the zener diode have?
2. If the impurity concentration is increased, how the depletion width effected?
3. Does the dynamic impendence of a zener diode vary?
4. Explain briefly about avalanche and zener breakdowns?
5. Draw the zener equivalent circuit?
6. Differentiate between line regulation & load regulation?
7. In which region zener diode can be used as a regulator?

8. How the breakdown voltage of a particular diode can be controlled?
9. What type of temperature coefficient does the Avalanche breakdown has?
10. By what type of charge carriers the current flows in zener and avalanche breakdown diodes?

OBSERVATIONS:

A) Static characteristics:

S.NO	Applied Voltage(V)	Forward Voltage (V_f)	Forward Current I_f (mA)

B) Reverse Characteristics:

S.NO	Applied Voltage(V)	Reverse Voltage(V_r)	Reverse Current I_r (mA)

RESULT:

9. HALF WAVE RECTIFIER WITH AND WITHOUT FILTERS

AIM: To examine the input and output waveforms of half wave Rectifier and also calculate ripple factor.

1. with Filter
2. without Filter

APPARATUS:

Digital Multimeter	- 1No.
Transformer (6V-0-6V)	- 1No.
Diode, 1N4007	- 1No.
Capacitor 100µf/470 µf	- 1No.
Decade Resistance Box	-1No.
Breadboard	
CRO and CRO probes	
Connecting wires	

THEORY:

In Half Wave Rectification, When AC supply is applied at the input, only Positive Half Cycle appears across the load whereas, the negative Half Cycle is suppressed. How this can be explained as follows:

During positive half-cycle of the input voltage, the diode D1 is in forward bias and conducts through the load resistor R_L. Hence the current produces an output voltage across the load resistor R_L, which has the same shape as the +ve half cycle of the input voltage.

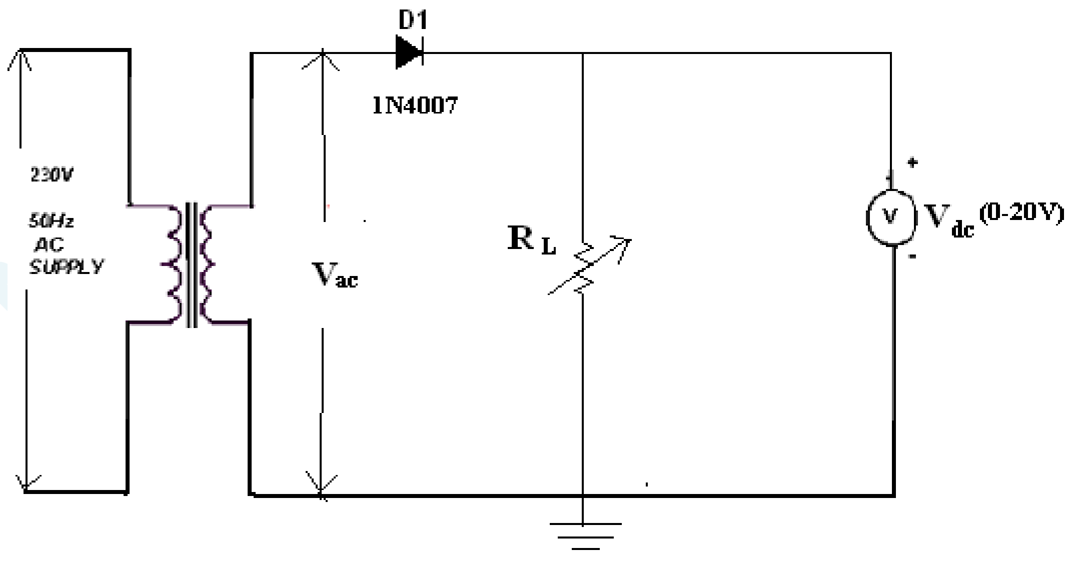
During the negative half-cycle of the input voltage, the diode is reverse biased and there is no current through the circuit. i.e., the voltage across R_L is zero. The net result is that only the +ve half cycle of the input voltage appears across the load. The average value of the half wave rectified o/p voltage is the value measured on dc voltmeter.

For practical circuits, transformer coupling is usually provided for two reasons.

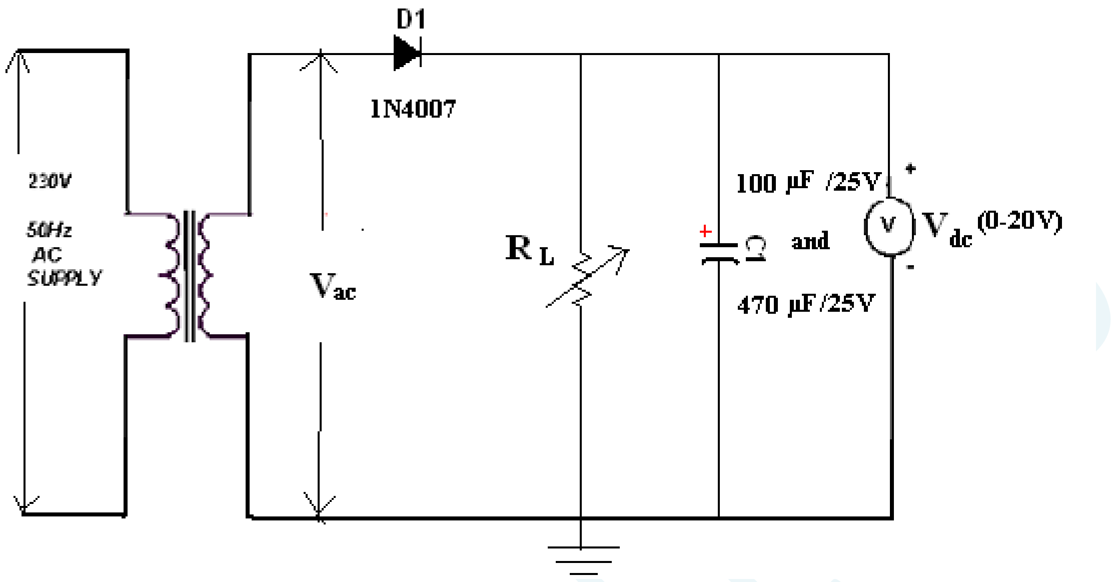
1. The voltage can be stepped-up or stepped-down, as needed.
2. The ac source is electrically isolated from the rectifier. Thus preventing shock hazards in the secondary circuit. The efficiency of the Half Wave Rectifier is 40.6%

CIRCUIT DIAGRAM:

A) Half wave Rectifier without filter:



B) Half wave Rectifier with filter



PROCEDURE:

1. Connections are made as per the circuit diagram.
2. Connect the primary side of the transformer to ac mains and the secondary side to the rectifier input.
3. By the multimeter, measure the ac input voltage of the rectifier and, ac and dc voltage at the output of the rectifier.
4. Find the theoretical of dc voltage by using the formula,

$$V_{dc}=V_m/\Pi$$

Where, $V_m=2V_{rms}$, (V_{rms}=output ac voltage.)

5. The Ripple factor is calculated by using the formula r = ac output voltage/dc output voltage.

Theoretical calculations for Ripple factor:

Without Filter:

$V_{rms}=V_m/2$

$V_m=2V_{rms}$

$V_{dc}=V_m/\Pi$

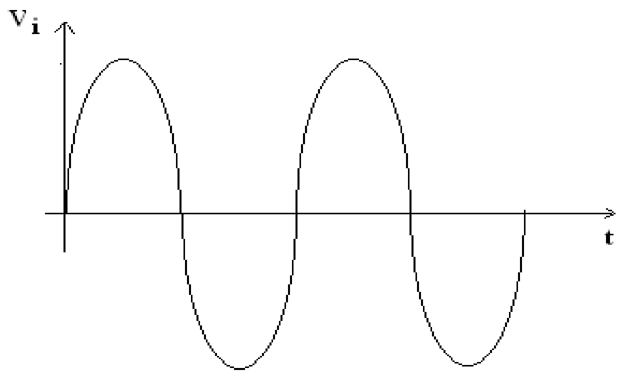
Ripple factor $r=\sqrt{(V_{rms}/V_{dc})^2-1}=1.21$

With Filter:

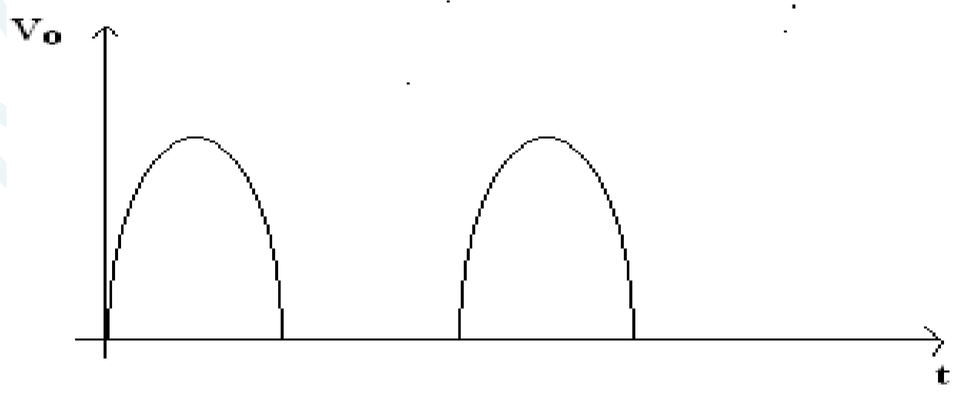
Ripple factor, $r=1/(2\sqrt{3}\,f\,C\,R)$

MODEL WAVEFORMS:

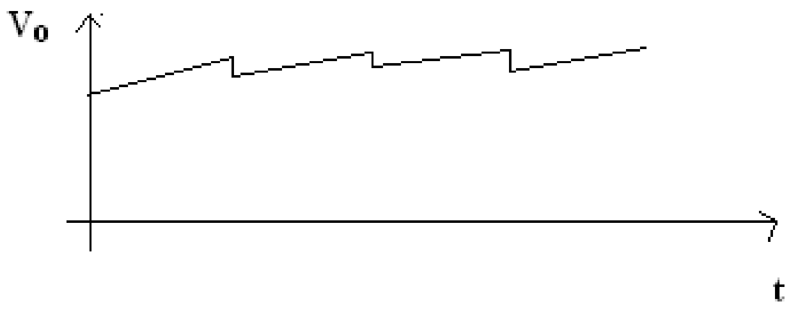
A) INPUT WAVEFORM



B) OUTPUT WAVFORM WITHOUT FILTER



C) OUTPUT WAVEFORM WITH FILTER:



PRECAUTIONS:

1. The primary and secondary side of the transformer should be carefully identified
2. The polarities of all the diodes should be carefully identified.

3. While determining the % regulation, first Full load should be applied and then it should be decremented in steps.

VIVA QUESTIONS:

1. What is the PIV of Half wave rectifier?
2. What is the efficiency of half wave rectifier?
3. What is the rectifier?
4. What is the difference between the half wave rectifier and full wave Rectifier?
5. What is the o/p frequency of Bridge Rectifier?
6. What are the ripples?
7. What is the function of the filters?
8. What is TUF?
9. What is the average value of o/p voltage for HWR?
10. What is the peak factor?

10. FULL WAVE RECTIFIER WITH AND WITHOUT FILTERS

AIM:

To Examine the input and output waveforms of Full Wave Rectifier and also calculate its load regulation and ripple factor.

1. with Filter
2. without Filter

APPARATUS:

Digital multimetersMultimeter	- 1No.
Transformer (6V-0-6V)	- 1No.
Diode, 1N4007	- 2No.
Capacitor 100μf/470 μf	- 1No.
Decade Resistance Box	-1No.
Bread board	
CRO and CRO probes	
Connecting wires	

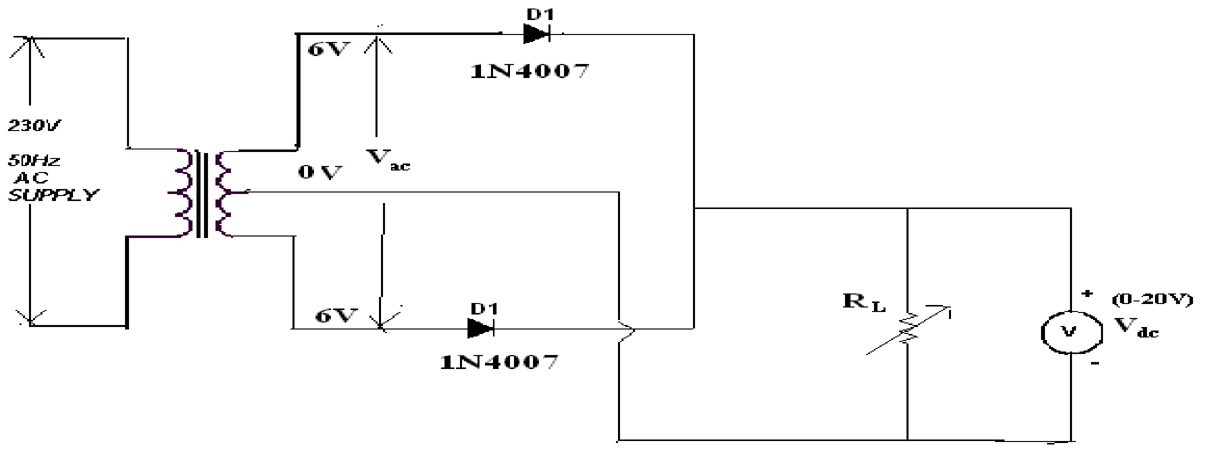
THEORY:

The circuit of a center-tapped full wave rectifier uses two diodes D1&D2. During positive half cycle of secondary voltage (input voltage), the diode D1 is forward biased and D2is reverse biased. So the diode D1 conducts and current flows through load resistor R_L.

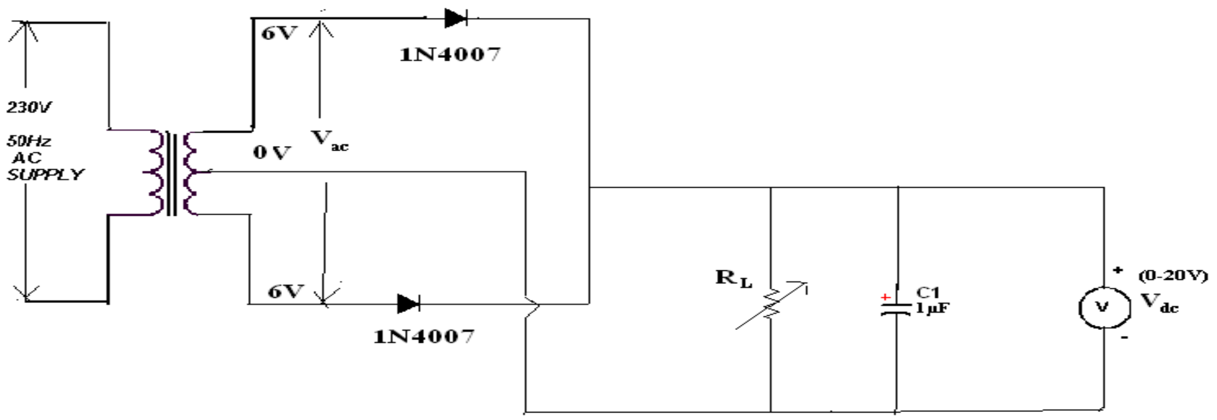
During negative half cycle, diode D2 becomes forward biased and D1 reverse biased. Now, D2 conducts and current flows through the load resistor R_L in the same direction. There is a continuous current flow through the load resistor R_L, during both the half cycles and will get unidirectional current as show in the model graph. The difference between full wave and half wave rectification is that a full wave rectifier allows unidirectional (one way) current to the load during the entire 360 degrees of the input signal and half-wave rectifier allows this only during one half cycle (180 degree).

CIRCUIT DIAGRAM:

A) FULL WAVE RECTIFIER WITHOUT FILTER:



B) FULL WAVE RECTIFIER WITH FILTER:



PROCEDURE:

1. Connections are made as per the circuit diagram.
2. Connect the ac mains to the primary side of the transformer and the secondary side to the rectifier.
3. Measure the ac voltage at the input side of the rectifier.
4. Measure both ac and dc voltages at the output side the rectifier.
5. Find the theoretical value of the dc voltage by using the formula Vdc=2Vm/Π
6. Connect the filter capacitor across the load resistor and measure the values of Vac and Vdc at the output.
7. The theoretical values of Ripple factors with and without capacitor are calculated.
8. From the values of Vac and Vdc practical values of Ripple factors are calculated. The practical values are compared with theoretical values.

THEORITICAL CALCULATIONS:

$$V_{rms} = V_m / \sqrt{2}$$
$$V_m = V_{rms}\sqrt{2}$$
$$V_{dc} = 2V_m/\Pi$$

(**i)Without filter**:

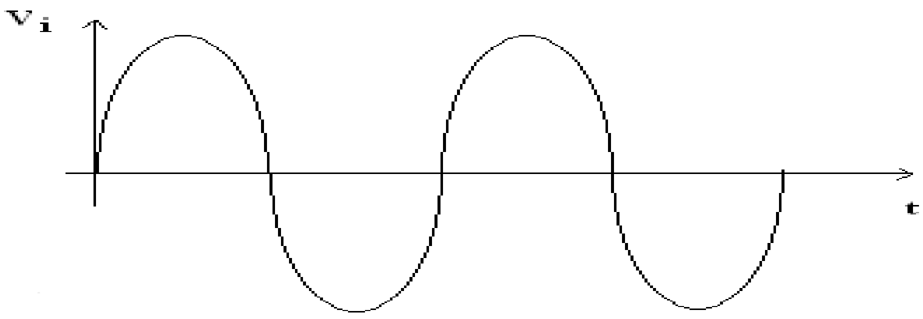
$$\text{Ripple factor, } r = \sqrt{(V_{rms}/V_{dc})^2 - 1} = 0.812$$

(ii)With filter:

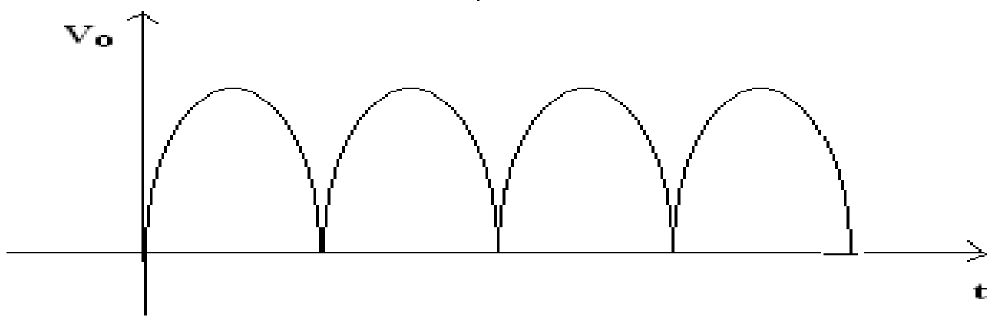
$$\text{Ripple factor, } r = 1/(4\sqrt{3}\, f\, C\, R_L)$$

MODEL WAVEFORMS:

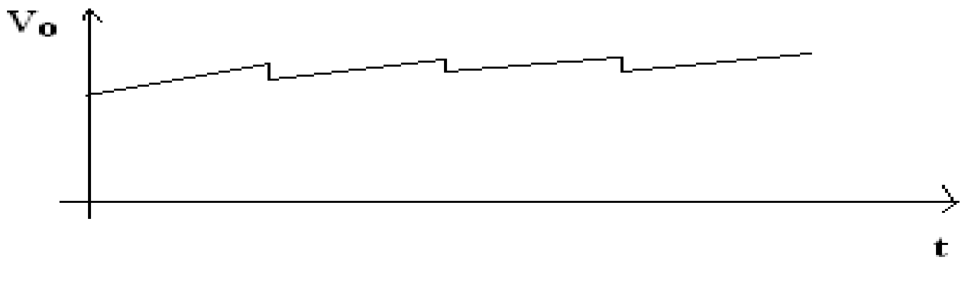
A) INPUT WAVEFORM



B) OUTPUT WAVEFORM WITHOUT FILTER:



C) OUTPUT WAVEFORM WITHOUT FILTER:



PRECAUTIONS:

1. The primary and secondary side of the transformer should be carefully identified.
2. The polarities of all the diodes should be carefully identified.

VIVA QUESTIONS:

1. Define regulation of the full wave rectifier?
2. Define peak inverse voltage (PIV)? And write its value for Full-wave rectifier?
3. If one of the diode is changed in its polarities what wave form would you get?
4. Does the process of rectification alter the frequency of the waveform?
5. What is ripple factor of the Full-wave rectifier?
6. What is the necessity of the transformer in the rectifier circuit?
7. What are the applications of a rectifier?
8. What is meant by ripple and define Ripple factor?
9. Explain how capacitor helps to improve the ripple factor?
10. Can a rectifier made in INDIA (V=230v, f=50Hz) be used in USA (V=110v, f=60Hz)?

RESULT

12.INPUT AND OUTPUT CHARACTERISTICS OF TRANSISTOR IN CE CONFIGARATION

AIM:

1. To draw the input and output characteristics of transistor connected in CE configuration
2. To find β of the given transistor and also its input and output Resistances

APPARATUS:

Transistor, BC107	-1No.
Regulated power supply (0-30V)	-1No.
Voltmeter (0-20V)	- 2No.
Ammeters (0-20mA)	-1No.
Ammeters (0-200μA)	-1No.
Resistor- 100Ω	-1No
Resistor-1KΩ	-1No.
Bread board	
Connecting wires	

THEORY:

In common emitter configuration, input voltage is applied between base and emitter terminals and out put is taken across the collector and emitter terminals. Therefore the emitter terminal is common to both input and output.

The input characteristics resemble that of a forward biased diode curve. This is expected since the Base-Emitter junction of the transistor is forward biased. As compared to CB arrangement I_B increases less rapidly with $V_{BE.}$ Therefore input resistance of CE circuit is higher than that of CB circuit.

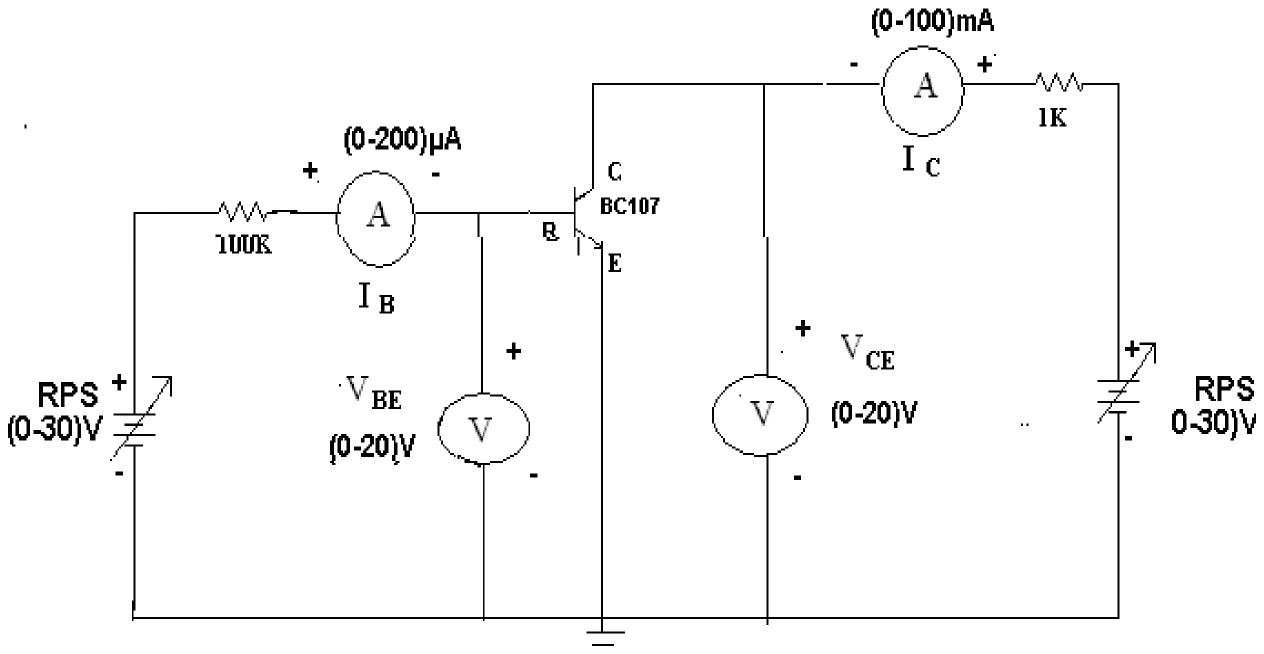
The output characteristics are drawn between I_c and V_{CE} at constant $I_{B.}$ the collector current varies with V_{CE} upto few volts only. After this the collector current becomes almost constant, and independent of $V_{CE.}$ The value of V_{CE} up to which the collector current changes with V $_{CE}$ is known as Knee voltage. The transistor always operated in the region above Knee voltage, $I_{C\ is}$ always constant and is approximately equal to $I_{B.}$The current amplification factor of CE configuration is given by

$$\beta = \Delta I_C/\Delta I_B$$

Input Resistance, r_i = $\Delta \mathbf{V_{BE}} / \Delta \mathbf{I_B}$ (μA) at Constant V_{CE}

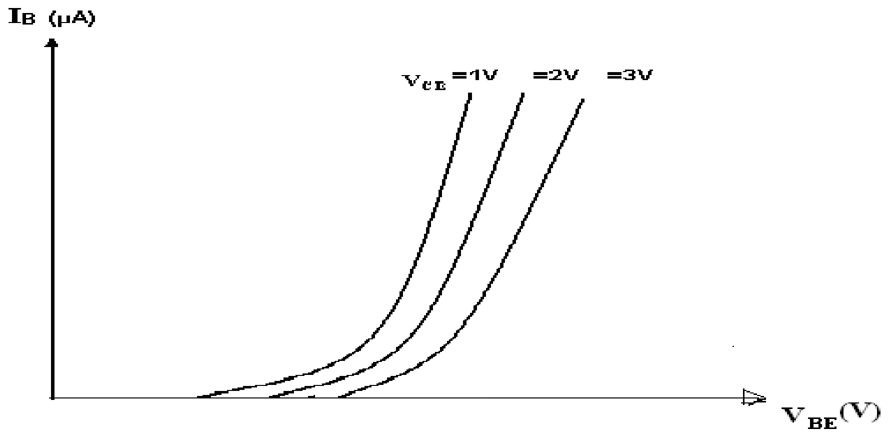
Output Résistance, r_o = $\Delta \mathbf{V_{CE}} / \Delta \mathbf{I_C}$ at Constant I_B (μA)

CIRCUIT DIAGRAM:

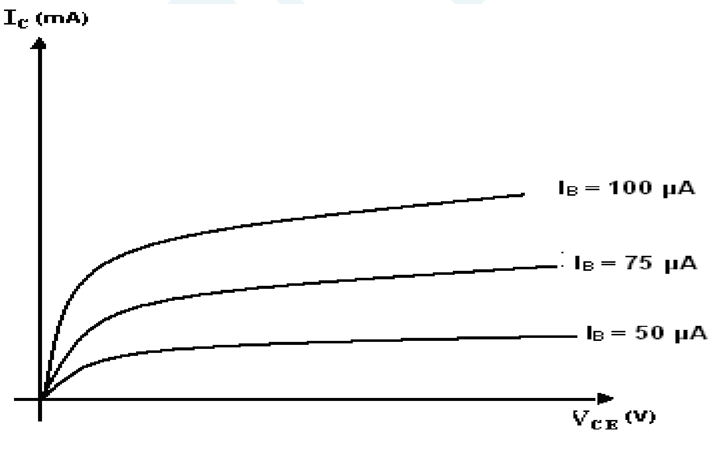


MODEL GRAPHS:

A) INPUT CHARACTERISTICS:



B) OUTPUT CHARACTERSITICS:



PROCEDURE:

A) **INPUT CHARECTERSTICS:**

1. Connect the circuit as per the circuit diagram.
2. For plotting the input characteristics the output voltage V_{CE} is kept constant at 1V and for different values of V_{BB} , note down the values of I_B and V_{BE}
3. Repeat the above step by keeping V_{CE} at 2V and 4V and tabulate all the readings.
4. plot the graph between V_{BE} and I_B for constant V_{CE}

B) **OUTPUT CHARACTERSTICS:**

1. Connect the circuit as per the circuit diagram
2. for plotting the output characteristics the input current I_B is kept constant at 50μA and for different values of V_{CC} note down the values of I_C and V_{CE}
3. Repeat the above step by keeping I_B at 75 μA and 100 μA and tabulate the all the readings
4. plot the graph between V_{CE} and I_C for constant I_B

PRECAUTIONS:

1. The supply voltage should not exceed the rating of the transistor
2. Meters should be connected properly according to their polarities

VIVA QUESTIONS:

1. What is the range of β for the transistor?
2. What are the input and output impedances of CE configuration?
3. Identify various regions in the output characteristics?
4. What is the relation between α and β?
5. Define current gain in CE configuration?

OBSERVATIONS:

A) INPUT CHARACTERISTICS:

V_{BB}	V_{CE} = 1V		V_{CE} = 2V		V_{CE} = 4V	
	V_{BE}(V)	I_B(μA)	V_{BE}(V)	I_B(μA)	V_{BE}(V)	I_B(μA)

B) OUTPUT CHAREACTARISTICS:

S.NO	I_B = 50 µA		I_B = 75 µA		I_B = 100 µA	
	V_{CE}(V)	I_C(mA)	V_{CE}(V)	I_C(mA)	V_{CE}(V)	I_C(mA)

RESULT:

Input & Output Characteristics of CB Configuration

Aim:

To study the input and output characteristics of a transistor in Common Base Configuration.

Components:

S.No.	**Name**	**Quantity**
1	Transistor BC 107	1(One) No.
2	Resistors (1KΩ)	2(Two) No.
3	Bread board	1(One) No.

Equipment

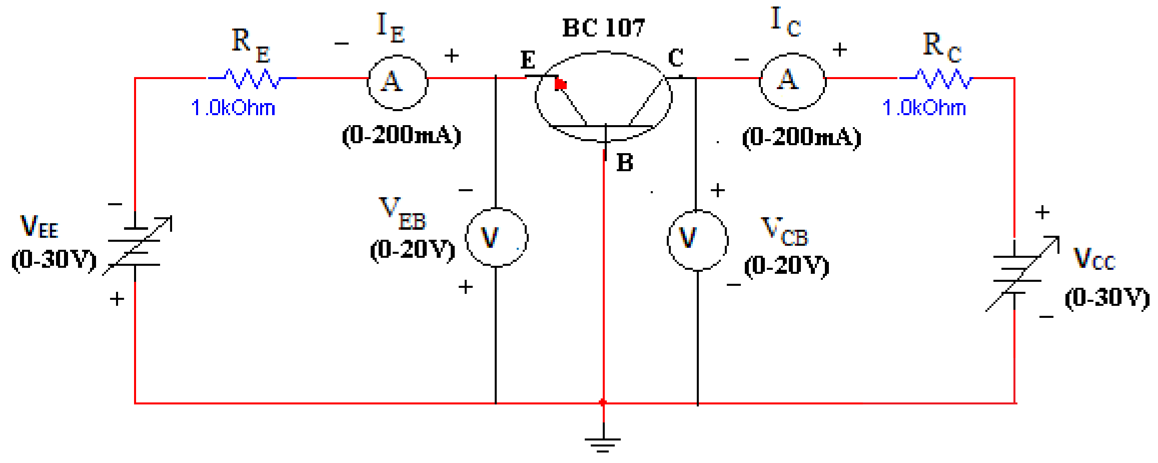
S.No.	**Name**	**Quantity**
1	Dual DC Regulated Power supply (0 – 30 V)	1(One) No.
2	Digital Ammeters (0 – 200 mA)	2(Two) No.
3	Digital Voltmeter (0-20V)	2(Two) No.
4	Connecting wires (Single Strand)	2

Specifications:

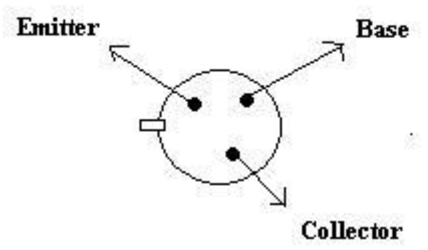
For Transistor BC 107:

- Max Collector Current = 0.1A
- V_{ceo} max = 50V

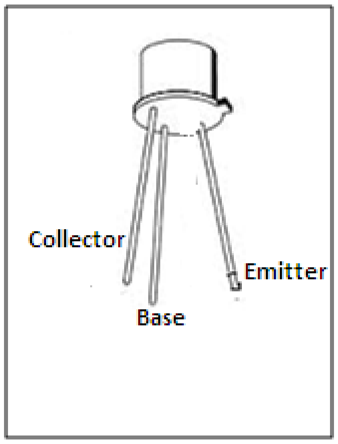
Circuit Diagram:



Pin assignment of Transistor:



View from side of pins



View from top of casing

Operation:

Bipolar Junction Transistor (BJT) is a three terminal (emitter, base, collector) semiconductor device. There are two types of BJTs, namely NPN and PNP. It consists of two PN junctions, namely emitter junction and collector junction.

The basic circuit diagram for studying input characteristics is shown in the circuit diagram. The input is applied between emitter and base, the output is taken between collector and base. Here base of the transistor is common to both input and output and hence the name is Common Base Configuration.

Input characteristics are obtained between the input current and input voltage at constant output voltage. It is plotted between $\mathbf{V}_{EE}$ and $\mathbf{I}_E$ at constant $\mathbf{V}_{CB}$ in CB configuration.

Output characteristics are obtained between the output voltage and output current at constant input current. It is plotted between $\mathbf{V}_{CB}$ and $\mathbf{I}_C$ at constant $\mathbf{I}_E$ in CB configuration.

Procedure:

Input Characteristics:

1. Connect the circuit as shown in the circuit diagram.
2. Keep output voltage V_{CB} = 0V by varying V_{CC}.
3. Varying V_{EE} gradually, note down emitter current I_E and emitter-base voltage(V_{EE}).
4. Step size is not fixed because of nonlinear curve. Initially vary V_{EE} in steps of 0.1 V. Once the current starts increasing vary V_{EE} in steps of 1V up to 12V.
5. Repeat above procedure (step 3) for V_{CB} = 4V.

Output Characteristics:

1. Connect the circuit as shown in the circuit diagram.
2. Keep emitter current I_E = 5mA by varying V_{EE}.
3. Varying V_{CC} gradually in steps of 1V up to 12V and note down collector current I_C and collector-base voltage(V_{CB}).
4. Repeat above procedure (step 3) for I_E = 10mA.

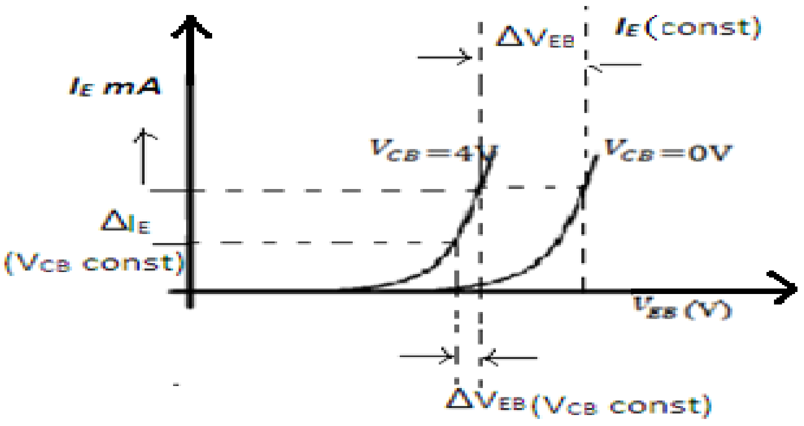
Repeat above procedure (step 3) for I_E = 10mA.

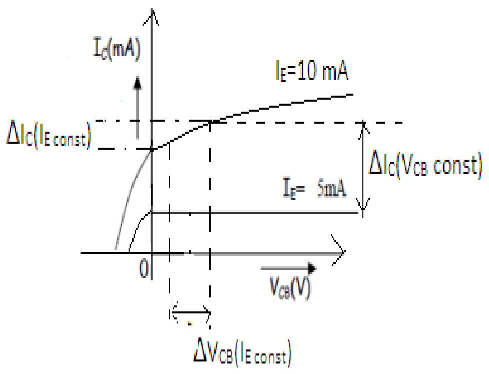
Observations:

Input Characteristics				
V_{EE} (Volts)	V_{CB} = 0V		V_{CB} = 4V	
	V_{EB} (Volts)	I_E (mA)	V_{EB} (Volts)	I_E (mA)

Output Characteristics						
V_{CC} (Volts)	I_E = 0mA		I_E = 5V		IE = 10mA	
	V_{CB} (Volts)	I_C (mA)	V_{CB} (Volts)	I_C (mA)	V_{CB} (Volts)	I_C (mA)

Graph:





1. Plot the input characteristics for different values of $\mathbf{V_{CB}}$ by taking $\mathbf{V_{EE}}$ on X-axis and $\mathbf{I_E}$ on Y-axis taking $\mathbf{V_{CB}}$ as constant parameter.
2. Plot the output characteristics by taking $\mathbf{V_{CB}}$ on X-axis and taking $\mathbf{I_C}$ on Y-axis taking $\mathbf{I_E}$ as a constant parameter.

Calculations from Graph:

The h-parameters are to be calculated from the following formulae:

1. **Input Characteristics:** To obtain input resistance, find ΔV_{EE} and ΔI_E for a constant V_{CB} on one of the input characteristics.

   Input impedance = $h_{ib} = R_i = \Delta V_{EE} / \Delta I_E$ (V_{CB} = constant)

   Reverse voltage gain = hrb = $\Delta V_{EB} / \Delta V_{CB}$ (I_E = constant)

2. **Output Characteristics:** To obtain output resistance, find ΔI_C and ΔV_{CB} at a constant I_E.

   Output admitance = $h_{ob} = 1/Ro = \Delta I_C / \Delta V_{CB}$ (I_E = constant)

   Forward current gain = $h_{fb} = \Delta I_C / \Delta I_E$ (V_{CB} = constant)

Inference:

1. Input resistance is in the order of tens of ohms since Emitter-Base Junction is forward biased.
2. Output resistance is in order of hundreds of kilo-ohms since Collector-Base Junction is reverse biased.
3. Higher is the value of V_{CB}, smaller is the cut in voltage.
4. Increase in the value of I_B causes saturation of transistor at small voltages.

Precautions:

1. While performing the experiment do not exceed the ratings of the transistor. This may lead to damage the transistor.
2. Connect voltmeter and ammeter in correct polarities as shown in the circuit diagram.
3. Do not switch ON the power supply unless you have checked the circuit connections as per the circuit diagram.
4. Make sure while selecting the emitter, base and collector terminals of the transistor.

Result:

Discussion/Viva Questions:

1. What is transistor?

Ans: A transistor is a semiconductor device used to amplify and switch electronic signals and electrical power. It is composed of semiconductor material with at least three terminals for connection to an external circuit. The term transistor was coined by John R. Pierce as a portmanteau of the term "transfer resistor".

2. Write the relation between α and β?

Ans: $\beta_F = \frac{\alpha_F}{1-\alpha_F} \Leftrightarrow \alpha_F = \frac{\beta_F}{\beta_F + 1}$

3. Define α (alpha)? What is the range of α?

Ans: The important parameter is the common-base current gain, α. The common-base current gain is approximately the gain of current from emitter to collector in the forward-active region. This ratio usually has a value close to unity; between 0.98 and 0.998.

4. Why α is less than unity?

Ans: It is less than unity due to recombination of charge carriers as they cross the base region.