Ex. No. 1a **Simple Calculator**
Date:

Aim

To implement a simple calculator using switch case statement.

Algorithm

1. **Start**
2. **Display calculator menu**
3. **Read the *operator* symbol and operands *n1*, *n2***
4. **If *operator* = + then**
   calculate *result* = *n1* + *n2*
   Else if *operator* = – then
   calculate *result* = *n1* – *n2*
   Else if *operator* = * then
   calculate *result* = *n1* * *n2*
   Else if *operator* = / then
   calculate *result* = *n1* / *n2*
   Else if *operator* = % then
   calculate *result* = *n1* % *n2*
   Else
   print "Invalid operator"
5. **Print *result***
6. **Stop**

Program

```
/* Simple Calculator using switch */

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

main()
{
   int n1, n2, result;
   char op;
   clrscr();
   printf("\n Simple Calculator");
   printf("\n + Summation");
   printf("\n - Difference");
   printf("\n * Product");
   printf("\n / Quotient");
   printf("\n % Remainder");
   printf("\n Enter the operator : ");
   op = getchar();
   printf("Enter operand1 and operand2 : ");
   scanf("%d%d",&n1,&n2);

   switch(op)
   {
      case '+':
         result = n1 +n2;
         break;
      case '-':
         result = n1 - n2;
         break;
      case '*':
         result = n1 * n2;
         break;
      case '/':
         result = n1 / n2;
         break;
      case '%':
         result = n1 % n2;
         break;
      default:
         printf("Invalid operator");
         exit(-1);
   }
   printf("%d %c %d = %d", n1, op, n2, result);
   getch();
}
```

Output

```
Simple Calculator
 + Summation
 - Difference
 * Product
 / Quotient
 % Remainder
Enter the operator : -
Enter operand1 and operand2 : 2 4
2 - 4 = -2


Simple Calculator
 + Summation
 - Difference
 * Product
 / Quotient
 % Remainder
Enter the operator : %
Enter operand1 and operand2 : 5 2
5 % 2 = 1
```

Result

Thus simple calculator functionality was executed using menu-oriented approach.

Ex. No. 1b **Armstrong Numbers**
Date:

Aim

To generate Armstrong numbers from 1- 1000 using nested loop.

Algorithm

1. **Start**
2. **Loop *i* to generate numbers from 1 to 1000**
3. **Initialize *sum* to 0.**
4. **Extract each digit of *i*, say *d***
5. **Cube the digit and add it to *sum***
6. **If sum = *i* then print *i***
7. **Stop**

Program

```
/* Armstrong Numbers 1 - 1000 */

#include <stdio.h>
#include <conio.h>

main()
{
   int i, n, d, sum;
   clrscr();

   printf("Armstrong numbers : ");
   for(i=2; i<=1000; i++)
   {
      sum = 0;
      n = i;
      while(n)
      {
         d = n % 10;
         sum = sum + (d * d * d);
         n = n / 10;
      }
      if (sum == i)
         printf("%d  ", i);
   }
   getch();
}
```

Output

```
Armstrong numbers : 153  370  371  407
```

Result

Thus first set of armstrong numbers is displayed using nested loop.

Ex. No. 1c **Sum of Digits**
Date:

Aim

To find the sum of the digits of a given number using while statement. .

Algorithm

1. **Start**
2. **Read *num***
3. **Initialize *sum* to 0.**
4. **Repeat until *num* = 0**
   Obtain last digit *d* = *num* % 10
   Add *d* to *sum*
   ***num* = *num* / 10**
5. **Print *sum***
6. **Stop**

Program

```
/* Sum of digits in a given number */

#include <stdio.h>
#include <conio.h>

main()
{
   int n, d, sum;
   clrscr();
   printf("Enter a number : ");
   scanf("%d", &n);

   sum = 0;
   while(n)
   {
      d = n % 10;
      sum = sum + d;
      n = n / 10;
   }

   printf("Sum of digits : %d", sum);
   getch();
}
```

Output

```
Enter a number : 58349
Sum of digits : 29
```

Result

Thus digits of the given number were summed up.

Ex. No. 1d First N numbers
Date:

Aim

To print first N numbers divisible by 3 using for loop.

Algorithm

1. **Start**
2. **Loop *i* to generate numbers**
3. **If *i* is divisible by 3 then print *i***
4. **Stop**

Program

```
/* First N numbers divisible by 3 */

#include <stdio.h>
#include <conio.h>

main()
{
     int i=0, j, n;
     clrscr();
     printf("\n Enter value for n : ");
     scanf("%d", &n);

     printf("First N numbers divisible by 3:");
     for(j=1; ;j++)
     {
          if (j%3 == 0)
          {
               i++;
               printf("%d  ", j);
          }
          if (i == n)
               break;
     }
     getch();
}
```

Output

```
Enter value for n : 8
First N numbers divisible by 3:
3  6  9  12  15  18  21  24
```

Result

Thus first ‘n’ numbers divisible by 3 was generated successfully.

Ex. No. 1e **Factorial Value**
Date:

Aim

To find the factorial of a number using function.

Algorithm

1. **Start**
2. **Read the value of *n***
3. **Call function *factorial*(*n*)**
4. **Print return value**
5. **Stop**

Function *factorial* (*n*)

i. **Initialize *f* to 1**
ii. **Initialize *i* to n**
iii. **Compute *f = f * i***
iv. **Decrement *i***
v. **Repeat steps *iii* and *iv* until *i* ≥ 1**
vi. **Return *f***

Program

```
/* Factorial using function */

#include <stdio.h>
#include <conio.h>

long factorial(int);

main()
{
     int n;
     long f;
     clrscr();
     printf("Enter a number : ");
     scanf("%d", &n);
     f = factorial(n);
     printf("Factorial value : %ld", f);
     getch();
}

long factorial(int n)
{
     int i;
     long f = 1;
     for(i=n; i>=1; i--)
          f = f * i;
     return f;
}
```

Output

```
Enter a number : 6
Factorial value : 720

Enter a number : 12
Factorial value : 479001600
```

Result

Thus factorial value of a given number was obtained using function.

Ex. No. 1f **Array Maximum**
Date:

Aim

To find the greatest of 'N' numbers stored in an array.

Algorithm

1. **Start**
2. **Read number of array elements as *n***
3. **Read array elements A_i, $i = 0,1,2,\ldots n-1$**
4. **Assume first element A_0 to be *max***
5. **Compare each array element A_i with *max***
   If $max < A_i$ then $max = A_i$
6. **Print *max***
7. **Stop**

Program

```
/* Maximum of an array */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[10];
   int i, max, n;
   clrscr();
   printf("Enter number of elements : ");
   scanf("%d", &n);

   printf("Enter Array Elements \n");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   max = a[0];
   for(i=1; i<n; i++)
   {
      if (max < a[i])
         max = a[i];
   }

   printf("Maximum value = %d ", max);
   getch();
}
```

Output

```
Enter number of elements : 6
Enter Array Elements
3
8
-7
11
-9
0
Maximum value = 11
```

Result

Thus maximum element of an array was determined.

Ex. No. 1g **Matrix Addition**
Date:

Aim

To add the given matrices using function.

Algorithm

1. **Start**
2. **Read the order of matrices as *m* and *n***
3. **Read matrix *A* elements A_{ij}**
4. **Read matrix *B* elements B_{ij}**
5. **Compute resultant matrix by adding corresponding elements**
   $C_{ij} = A_{ij} + B_{ij}$
6. **Print matrix C_{ij},**
7. **Stop**

Program

```
/* Matrix addition using function */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[5][5], b[5][5], c[5][5];
   int i, j, row, col;
   clrscr();

   printf("\nEnter order for matrix : ");
   scanf("%d%d", &row, &col);

   printf("\nEnter elements for A matrix\n");
   for(i=0; i<row; i++)
      for(j=0; j<col; j++)
         scanf("%d", &a[i][j]);

   printf("\nEnter elements for B matrix\n");
   for(i=0; i<row; i++)
      for(j=0; j<col; j++)
         scanf("%d", &b[i][j]);

   for(i=0; i<row; i++)
      for(j=0; j<col; j++)
         c[i][j] = a[i][j] + b[i][j];

   printf("\n Contents of C matrix \n");
   for(i=0; i<row; i++)
   {
      for(j=0; j<col; j++)
      {
         printf("%3d", c[i][j]);
      }
      printf("\n");
   }
   getch();
}
```

Output

```
Enter order for matrix : 2 2

Enter elements for A matrix
1 1
1 1

Enter elements for B matrix
2 2
3 4

Contents of C matrix
  3  3
  4  5
```

Result

Thus the given matrices were added.

Ex. No. 2a Palindrome
Date:

Aim

To determine whether the input string is palindrome using string handling functions.

Algorithm

1. **Start**
2. **Read the string, say *str***
3. **Copy *str*onto *rev*using `strcpy` function**
4. **Reverse *rev* using `strrev` function**
5. **Compare *str* and *rev*using `strcmp` function**
6. **If outcome = 0 then**
   Print "Given string is palindrome"
   Else
   Print "Given string is not palindrome"
7. **Stop**

Program

```
/* Palindrome using string functions */

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

main()
{
   charstr[40], rev[40];
   int x;
   clrscr();
   printf("Enter the string: ");
   scanf("%s", str);

   strcpy(rev, str);
   strrev(rev);
   printf("Reversed string is: %s \n",rev);

   x = strcmpi(str, rev);
   if (x == 0)
      printf("Given string is a palindrome");
   else
      printf("Given string is not a palindrome");

   getch();
}
```

Output

```
Enter the string:malayalam
Reversed string is:malayalam
Given string is a palindrome

Enter the string: Computer
Reversed string is: retupmoC
Given string is not a palindrome
```

Result

Thus the given input string is checked for palindrome using string functions.

Ex. No. 2b **Alphabetical Ordering**
Date:

Aim

To sort the given set of names in alphabetical order.

Algorithm

1. **Start**
2. **Read number of name as *n***
3. **Set up a loop and read the name list in *name* array**
4. **Assign 0 to *i***
5. **Assign *i*+1 to *j***
6. **If i^{th} *name* and j^{th} *name* not in order, then swap the strings**
7. **Increment *j* by 1**
   Repeat steps 6 and 7 until *j* < *n*
8. **Increment *i* by 1**
   Repeat steps 5–8 until *i* < *n*-1
9. **Set up a loop and print the sorted name array**
10. **Stop**

Program

```
/* Alphabetical Order */

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

main()
{
   char name[20][15], t[15];
   int i, j, n;
   clrscr();
   printf("Enter number of students : ");
   scanf("%d", &n);
   printf("\nEnter student names\n");
   for(i=0; i<n; i++)
      scanf("%s", name[i]);

/* Bubble Sort method */
   for(i=0; i<n-1; i++)
   {
      for(j=i+1; j<n; j++)
      {
         if (strcmp(name[i],name[j]) > 0)
         {
            strcpy(t, name[i]);
            strcpy(name[i], name[j]);
            strcpy(name[j], t);
         }
      }
   }

   printf("\nAlphabetical Order \n");
   for(i=0; i<n; i++)
      printf("%s\n", name[i]);
   getch();
}
```

Output

```
Enter number of students : 10

Enter student names
Raghu
Praba
Gopal
Anand
Venkat
Kumar
Saravana
Naresh
Christo
Vasanth

Alphabetical Order
Anand
Christo
Gopal
Kumar
Naresh
Praba
Raghu
Saravana
Vasanth
Venkat
```

Result

Thus the given set of names were sorted as per alphabetical order

Ex. No. 3a **Pass By Value / Reference**
Date:

Aim

To demonstrate parameters passing to a function by reference.

Algorithm

1. **Start**
2. **Read values of a and b**
3. **Print *a* and *b***
4. **Call *swapref* function with address of *a* and *b***
5. **Print *a* and *b***
6. **Stop**

Function *swapref* (x, y)

i. **Assign value pointed by variable *x* to a temporary variable *t***
ii. **Assign value pointed by variable *y* to value pointed by variable *x***
iii. **Assign value t to value pointed by variable *y***

Function *swapval* (p, q)

i. **Assign value of *p* to a temporary variable *t***
ii. **Assign value of *q* to *p***
iii. **Assign value of *t* to *q***

Program

```
/* Pass by value and reference */

#include <stdio.h>
#include <conio.h>

void swapref(int *,int *);

main()
{
   int a, b;
   clrscr();
   printf("Enter value for A: ");
   scanf("%d", &a);
   printf("Enter value for B: ");
   scanf("%d", &b);

   swapref(&a, &b);
   printf("\n Values after Pass by Reference \n");
   printf("Value of A : %d \n", a);
   printf("Value of B : %d", b);

   swapval(a, b);
   printf("\n Values after Pass by Value \n");
   printf("Value of A : %d \n", a);
   printf("Value of B : %d", b);
   getch();
}

/* Pass by Value */
void swapval(int p, int q)
{
   int t;
   t = p;
   p = q;
   q = r;
}

/* Pass by Reference*/
void swapref(int *x, int *y)
{
   int t;
   t = *x;
   *x = *y;
   *y = t;
}
```

Output

```
Enter value for A : 10
Enter value for B : 20

Values after Pass by Reference
Value of A : 20
Value of B : 10

Values after Pass by Value
Value of A : 20
Value of B : 10
```

Result

Thus values of variables were exchanged by passing parameters by reference.

Ex. No. 3b **Payroll Application**
Date:

Aim

To generate employee payroll for an organization using structure.

Algorithm

1. **Start**
2. **Define *employee* structure with fields *empid, ename, basic, hra, da, it, gross* and *netpay***
3. **Read number of employees *n***
4. **Read *empid, ename,* and *basic* for *n* employees in an array of structure.**
5. **For each employee, compute**
   hra = 2% of basic
   da = 1% of basic
   gross = basic + hra + da
   it = 5% of basic
   netpay = gross - it
6. **Print *empid, ename, basic, hra, da, it, gross* and *netpay* for all employees**
7. **Stop**

Program

```
/* Payroll Generation */

#include <stdio.h>
#include <conio.h>

struct employee
{
   int empid;
   char name[15];
   int basic;
   float hra;
   float da;
   float it;
   float gross;
   float netpay;
};

main()
{
   struct employee emp[50];
   inti, j, n;

   clrscr();
   printf("\n Enter No. of Employees : ");
   scanf("%d", &n);

   for(i=0; i<n ;i++)
   {
      printf("\n Enter Employee Details\n");
      printf("Enter Employee Id   : ");
      scanf("%d", &emp[i].empid);
      printf("Enter Employee Name : ");
      scanf("%s", emp[i].ename);
      printf("Enter Basic Salary  : ");
      scanf("%d", &emp[i].basic);
   }

   for(i=0; i<n; i++)
   {
      emp[i].hra = 0.02 * emp[i].basic;
      emp[i].da  = 0.01 * emp[i].basic;
      emp[i].it  = 0.05 * emp[i].basic;
      emp[i].gross = emp[i].basic + emp[i].hra + emp[i].da;
      emp[i].netpay = emp[i].gross - emp[i].it;
   }
```

```
printf("\n\n\n\t\t\t\tXYZ& Co. Payroll\n\n");
   for(i=0;i<80;i++)
      printf("*");
   printf("EmpId\tName\t\tBasic\t HRA\t DA\t IT\t
Gross\t\tNet Pay\n");
   for(i=0;i<80;i++)
      printf("*");
   for(i=0; i<n; i++)
   {
      printf("\n%d\t%-15s\t%d\t%.2f\t%.2f\t%.2f\t%.2f\t%.2f",
         emp[i].empid, emp[i].ename, emp[i].basic, emp[i].hra,
         emp[i].da, emp[i].it, emp[i].gross, emp[i].netpay);
   }
   printf("\n");
   for(i=0;i<80;i++)
      printf("*");
   getch();
}
```

Output

```
Enter No. of Employees : 2

Enter Employee Details
Enter Employee Id   : 436
Enter Employee Name :Gopal
Enter Basic Salary  : 10000

Enter Employee Details
Enter Employee Id   : 463
Enter Employee Name :Rajesh
Enter Basic Salary  : 22000


                        XYZ & Co. Payroll

********************************************************************************
EmpId   Name            Basic    HRA     DA      IT      Gross          Net Pay
********************************************************************************

436     Gopal           10000   200.00  100.00  500.00  10300.00        9800.00
463     Rajesh          22000   440.00  220.00  1100.00 22660.00        21560.00

********************************************************************************
```

Result

Thus payroll for employees was generated using structure.

Ex. No.3c Cricket Team Stats

Date:

Aim

To define a structure for a cricket player and to print a team wise list containing names of players with their batting average.

Algorithm

1. **Start**
2. **Define *cricket* structure with fields *pcode, pname, tname,* and *bavg***
3. **Read number of players *n***
4. **Read details of each player in an array of structure.**
5. **Sort player structure according to team name.**
6. **Print players details**
7. **Stop**

Program

```
/* 3b - Cricket Statistics */

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

struct cricket
{
   int plcode;
   char name[15];
   char tname[15];
   float btavg;
};

main()
{
   struct cricket player[50], temp;
   int i, j, n;

   clrscr();
   printf("\n Enter No. of Players : ");
   scanf("%d", &n);

   for(i=0; i<n; i++)
   {
      printf("\nEnter Player Details\n");
      printf("Enter player code : ");
      scanf("%d", &player[i].plcode);
      printf("Enter player name : ");
      scanf("%s", player[i].name);
      printf("Enter team name : ");
      scanf("%s", player[i].tname);
      printf("Enter batting average : ");
      scanf("%f", &player[i].btavg);
   }

   for(i=0; i<n-1; i++)
   {
      for(j=i+1; j<n; j++)
      {
         if (strcmp(player[i].tname, player[j].tname) > 0)
         {
            temp = player[i];
            player[i] = player[j];
            player[j] = temp;
         }
      }
   }
```

```
   printf("\n\t PLAYER DETAILS-TEAM WISE \n");
   printf("\n P.Code \t");
   printf("%-15s %-15s", "Name", "Team");
   printf("Bat. Avg \n");
   for(i=0; i<n; i++)
   {
      printf("%d\t", player[i].plcode);
      printf("%-15s", player[i].name);
      printf("%-15s", player[i].tname);
      printf("%.2f\n", player[i].btavg);
   }
   getch();
}
```

Output

```
Enter No. of Players : 6

Enter Player Details
Enter player code : 23
Enter player name : Dhoni
Enter team name : CSK
Enter batting average : 45.23

Enter Player Details
Enter player code : 34
Enter player name : Maxwell
Enter team name : PW
Enter batting average : 67.2

Enter Player Details
Enter player code : 17
Enter player name : Raina
Enter team name : CSK
Enter batting average : 85

        PLAYER DETAILS-TEAM WISE
P.Code  Name           Team           Bat. Avg
17      Raina          CSK            85.00
23      Dhoni          CSK            45.23
34      Maxwell        PW             67.20
```

Result

Thus players details were sorted team-wise and printed.

Ex. No. 3d Complex Number Addition
Date:

Aim

To add two complex numbers using structures.

Algorithm

1. **Define structure for complex number with members real and imag**
2. **Read both the complex numbers**
3. **Add the corresponding real and imag of both complex numbers**
4. **Store the summation in a new complex number**
5. **Print the complex number**
6. **Stop**

Program

```
/* Addition of Complex numbers using structure */

#include <stdio.h>
#include <conio.h>

struct complex
{
   int real;
   int img;
};

main()
{
   struct complex c1, c2, c3;

   printf("Enter the real part for first complex number: ");
   scanf("%d", &c1.real);
   printf("Enter imaginary part for first complex number: ");
   scanf("%d", &c1.img);
   printf("First complex No.: %d + %di\n", c1.real, c1.img);

   printf("Enter the real part for second complex number: ");
   scanf("%d", &c2.real);
   printf("Enter imaginary part for second complex number: ");
   scanf("%d", &c2.img);
   printf("Second complex No.: %d + %di\n", c2.real, c2.img);

   c3.real = c1.real + c2.real;
   c3.img = c1.img + c2.img;
   printf("Sum of complex Nos.: %d + %di\n", c3.real, c3.img);
   getch();
}
```

Output

```
Enter the real part for first complex number: 3
Enter imaginary part for first complex number: 2
First complex No.: 3 + 2i
Enter the real part for second complex number: 4
Enter imaginary part for second complex number: 1
Second complex No.: 4 + 1i
Sum of the complex Nos.: 7 + 3i
```

Result

Thus the two complex numbers were added using structures.

Ex. No. 4a **Dynamic Memory Allocation**
Date:

Aim

To implement dynamic memory allocation of an one dimensional array.

Algorithm

1. **Start**
2. **Create pointer variables**
3. **Obtain number of elements say *n***
4. **Allocate memory dynamically for *n* elements using *malloc* function**
5. **Accept input for n elements using pointer notation**
6. **Determine *sum, min, max* for the given set of elements using pointer notation**
7. **Display *sum, min* and *max***
8. **Stop**

Program

```
/* Dynamic memory allocation */

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

main()
{
   int i, n, sum, min, max;
   int *arr, *p;

   printf("Enter number of elements : ");
   scanf("%d", &n);

   arr = (int *) malloc(n * sizeof(int));
   if(arr == NULL)
   {
      printf("Memory allocation  not feasible\n");
      exit(-1);
   }

   printf("Enter array elements: \n");
   for(p=arr; p<arr+n; p++)
      scanf("%d", p);

   sum = 0;
   min = max = *arr;
   for(p=arr; p<arr+n; p++)
   {
      if(min > *p)
         min = *p;
      if(max < *p)
         max = *p;
      sum = sum + *p;
   }

   printf("Sum of array : %d \n", sum);
   printf("Array minimum is : %d \n", min);
   printf("Array maximum is : %d \n", max);

   free(arr);
   getch();
}
```

Output

```
Enter number of elements : 5
Enter array elements:
12
8
16
5
7
Sum of array : 48
Array minimum is : 5
Array maximum is : 16
```

Result

Thus memory has been dynamically allocated and relinquished.

Ex. No. 4b **Singly Linked List**
Date:

Aim

To define a singly linked list node and perform operations such as insertions and deletions dynamically.

Algorithm

1. **Start**
2. **Define single linked list *node* as self referential structure**
3. **Create *Head* node with label = -1 and next = NULL using**
4. **Display menu on list operation**
5. **Accept user choice**
6. **If choice = 1 then**
   Locate node after which insertion is to be done
   Create a new node and get data part
   Insert new node at appropriate position by manipulating address
   Else if choice = 2
   Get node's data to be deleted.
   Locate the node and delink the node
   Rearrange the links
   Else
   Traverse the list from Head node to node which points to null
7. **Stop**

Program

```
/* Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>
#include <string.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch, fou=0;
   int k;
   struct node *h, *temp, *head, *h1;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->label = -1;
   head->next = NULL;

   while(-1)
   {
      clrscr();
      printf("\n\n SINGLY LINKED LIST OPERATIONS \n");
      printf("1->Add  ");
      printf("2->Delete  ");
      printf("3->View  ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         /* Add a node at any intermediate location */
         case 1:
            printf("\n Enter label after which to add : ");
            scanf("%d", &k);

            h = head;
            fou = 0;

            if (h->label == k)
               fou = 1;
```

```
   while(h->next != NULL)
   {
      if (h->label == k)
      {
           fou=1;
           break;
      }
      h = h->next;
   }
   if (h->label == k)
      fou = 1;

   if (fou != 1)
      printf("Node not found\n");
   else
   {
   temp=(struct node *)(malloc(sizeof(struct node)));
      printf("Enter label for new node : ");
      scanf("%d", &temp->label);
      temp->next = h->next;
      h->next = temp;
   }
   break;

/* Delete any intermediate node */
case 2:
   printf("Enter label of node to be deleted\n");
   scanf("%d", &k);
   fou = 0;
   h = h1 = head;
   while (h->next != NULL)
   {
      h = h->next;
      if (h->label == k)
      {
           fou = 1;
           break;
      }
   }

   if (fou == 0)
      printf("Sorry Node not found\n");
   else
      {
      while (h1->next != h)
           h1 = h1->next;
      h1->next = h->next;
      free(h);
      printf("Node deleted successfully \n");
   }
   break;
```

```
         case 3:
            printf("\n\n HEAD -> ");
            h=head;
            while (h->next != NULL)
            {
               h = h->next;
               printf("%d -> ",h->label);
            }
            printf("NULL");
            break;

         case 4:
            exit(0);
      }
   }
}
```

Output

```
SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label after which new node is to be added : -1
Enter label for new node : 23

SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label after which new node is to be added : 23
Enter label for new node : 67

SINGLY LINKED LIST OPERATIONS
1->Add  2->Delete  3->View  4->Exit
Enter your choice : 3
 HEAD -> 23 -> 67 -> NULL
```

Result

Thus operation on single linked list is performed.

Ex. No. 5a **Stack Array**
Date:

Aim

To implement stack operations using array.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 5**
3. **Initialize *top* = -1**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
   If top < max -1
   Increment top
   Store element at current position of top
   Else
   Print Stack overflow
   Else If choice = 2 then
   If top < 0 then
   Print Stack underflow
   Else
   Display current top element
   Decrement top
   Else If choice = 3 then
   Display stack elements starting from top
7. **Stop**

Program

```
/* Stack Operation using Arrays */

#include <stdio.h>
#include <conio.h>

#define max 5

static int stack[max];
int top = -1;

void push(int x)
{
   stack[++top] = x;
}

int pop()
{
   return (stack[top--]);
}

void view()
{
   int i;
   if (top < 0)
      printf("\n Stack Empty \n");
   else
   {
      printf("\n Top-->");
      for(i=top; i>=0; i--)
      {
         printf("%4d", stack[i]);
      }
      printf("\n");
   }
}

main()
{
   int ch=0, val;
   clrscr();

   while(ch != 4)
   {
      printf("\n STACK   OPERATION \n");
      printf("1.PUSH  ");
      printf("2.POP  ");
      printf("3.VIEW  ");
      printf("4.QUIT \n");
      printf("Enter Choice : ");
```

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

        switch(ch)
        {
           case 1:
              if(top < max-1)
              {
                 printf("\nEnter Stack element : ");
                 scanf("%d", &val);
                 push(val);
              }
              else
                 printf("\n Stack Overflow \n");
              break;
           case 2:
              if(top < 0)
                 printf("\n Stack Underflow \n");
              else
              {
                 val = pop();
                 printf("\n Popped element is %d\n", val);
              }
              break;
           case 3:
              view();
              break;
           case 4:
              exit(0);
           default:
              printf("\n Invalid Choice \n");
        }
     }
}
```

Output

```
STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 23
```

```
STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 34

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 1

Enter Stack element : 45

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 3

Top-->  45  34  23  12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 2

Popped element is 45

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 3

Top-->  34  23  12

STACK   OPERATION
1.PUSH  2.POP  3.VIEW  4.QUIT
Enter Choice : 4
```

Result

Thus push and pop operations of a stack was demonstrated using arrays.

Ex. No. 5b Queue Array
Date:

Aim

To implement queue operations using array.

Algorithm

1. **Start**
2. **Define a array *queue* of size *max* = 5**
3. **Initialize *front* = *rear* = –1**
4. **Display a menu listing queue operations**
5. **Accept choice**
6. **If choice = 1 then**
   If rear < max -1
   Increment rear
   Store element at current position of rear
   Else
   Print Queue Full
   Else If choice = 2 then
   If front = –1 then
   Print Queue empty
   Else
   Display current front element
   Increment front
   Else If choice = 3 then
   Display queue elements starting from front to rear.
7. **Stop**

Program

```
/* Queue Operation using Arrays */

#include <stdio.h>
#include <conio.h>

#define max 5

static int queue[max];
int front = -1;
int rear  = -1;

void insert(int x)
{
   queue[++rear] = x;
   if (front == -1)
      front = 0;
}

int remove()
{
   int val;
   val = queue[front];
   if (front==rear && rear==max-1)
      front = rear = -1;
   else
      front ++;
   return (val);
}

void view()
{
   int i;

   if (front == -1)
      printf("\n Queue Empty \n");
   else
   {
      printf("\n Front-->");
      for(i=front; i<=rear; i++)
         printf("%4d", queue[i]);
      printf("  <--Rear\n");
   }
}

main()
{
   int ch= 0,val;
   clrscr();
```

```
    while(ch != 4)
    {
       printf("\n QUEUE OPERATION \n");
       printf("1.INSERT  ");
       printf("2.DELETE  ");
       printf("3.VIEW  ");
       printf("4.QUIT\n");
       printf("Enter Choice : ");
       scanf("%d", &ch);

       switch(ch)
       {
          case 1:
             if(rear < max-1)
             {
                printf("\n Enter element to be inserted : ");
                scanf("%d", &val);
                insert(val);
             }
             else
                printf("\n Queue Full \n");
             break;
          case 2:
             if(front == -1)
                printf("\n Queue Empty \n");
             else
             {
                val = remove();
                printf("\n Element deleted : %d \n", val);
             }
             break;
          case 3:
             view();
             break;
          case 4:
             exit(0);
          default:
             printf("\n Invalid Choice \n");
       }
    }
}
```

Output

```
 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 12

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 23

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 34

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 45

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Enter element to be inserted : 56

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 1

Queue Full

 QUEUE OPERATION
1.INSERT  2.DELETE  3.VIEW  4.QUIT
Enter Choice : 3

Front-->  12  23  34  45  56  <--Rear
```

Result

Thus insert and delete operations of a queue was demonstrated using arrays.

Ex. No. 6a Stack Using Linked List
Date:

Aim

To implement stack operations using linked list.

Algorithm

1. **Start**
2. **Define a singly linked list node for stack**
3. **Create Head node**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
   Create a new node with data
   Make new node point to first node
   Make head node point to new node
   Else If choice = 2 then
   Make temp node point to first node
   Make head node point to next of temp node
   Release memory
   Else If choice = 3 then
   Display stack elements starting from head node till null
7. **Stop**

Program

```
/* Stack using Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch = 0;
   int k;
   struct node *h, *temp, *head;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->next = NULL;

   while(1)
   {
      printf("\n Stack using Linked List \n");
      printf("1->Push  ");
      printf("2->Pop  ");
      printf("3->View  ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            /* Create a new node */
            temp=(struct node *)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);
            h = head;
            temp->next = h->next;
            h->next = temp;
            break;

         case 2:
            /* Delink the first node */
            h = head->next;
            head->next = h->next;
```

```
            printf("Node %s deleted\n", h->label);
            free(h);
            break;

         case 3:
            printf("\n HEAD -> ");
            h = head;
            /* Loop till last node */
            while(h->next != NULL)
            {
               h = h->next;
               printf("%d -> ",h->label);
            }
            printf("NULL \n");
            break;

         case 4:
            exit(0);
      }
   }
}
```

Output

```
 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 23
New node added

 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 34

 Stack using Linked List
1->Push  2->Pop  3->View  4->Exit
Enter your choice : 3
HEAD -> 34 -> 23 -> NULL
```

Result

Thus push and pop operations of a stack was demonstrated using linked list.

Ex. No. 6b Queue Using Linked List
Date:

Aim

To implement queue operations using linked list.

Algorithm

1. **Start**
2. **Define a singly linked list node for stack**
3. **Create Head node**
4. **Display a menu listing stack operations**
5. **Accept choice**
6. **If choice = 1 then**
   Create a new node with data
   Make new node point to first node
   Make head node point to new node
   Else If choice = 2 then
   Make temp node point to first node
   Make head node point to next of temp node
   Release memory
   Else If choice = 3 then
   Display stack elements starting from head node till null
7. **Stop**

Program

```
/* Queue using Single Linked List */

#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <alloc.h>

struct node
{
   int label;
   struct node *next;
};

main()
{
   int ch=0;
   int k;
   struct node *h, *temp, *head;

   /* Head node construction */
   head = (struct node*) malloc(sizeof(struct node));
   head->next = NULL;

   while(1)
   {
      printf("\n Queue using Linked List \n");
      printf("1->Insert  ");
      printf("2->Delete  ");
      printf("3->View  ");
      printf("4->Exit \n");
      printf("Enter your choice : ");
      scanf("%d", &ch);

      switch(ch)
      {
         case 1:
            /* Create a new node */
            temp=(struct node *)(malloc(sizeof(struct node)));
            printf("Enter label for new node : ");
            scanf("%d", &temp->label);

            /* Reorganize the links */
            h = head;
            while (h->next != NULL)
               h = h->next;
            h->next = temp;
            temp->next = NULL;
            break;
```

```
         case 2:
            /* Delink the first node */
            h = head->next;
            head->next = h->next;
            printf("Node deleted \n");
            free(h);
            break;

         case 3:
            printf("\n\nHEAD -> ");
            h=head;
            while (h->next!=NULL)
            {
               h = h->next;
               printf("%d -> ",h->label);
            }
            printf("NULL \n");
            break;

         case 4:
            exit(0);
      }
   }
}
```

Output

```
 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 12

 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 1
Enter label for new node : 23

 Queue using Linked List
1->Insert  2->Delete  3->View  4->Exit
Enter your choice : 3
HEAD -> 12 -> 23 -> NULL
```

Result

Thus insert and delete operations of a stack was demonstrated using linked list.

Ex. No. 7a Infix To Postfix Conversion
Date:

Aim

To convert infix expression to its postfix form using stack operations.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 20**
3. **Initialize *top* = -1**
4. **Read the infix expression character-by-character**
   - **If character is an operand print it**
   - **If character is an operator**
     - **Compare the operator's priority with the stack[top] operator.**
     - **If the stack [top] has higher/equal priority than the input operator,**
       Pop it from the stack and print it.
     - **Else**
       Push the input operator onto the stack
   - **If character is a left parenthesis, then push it onto the stack.**
   - **If character is a right parenthesis, pop all operators from stack and print it until a left parenthesis is encountered. Do not print the parenthesis.**
   - **If character = $ then Pop out all operators, Print them and Stop**

.

Program

```
/* Conversion of infix to postfix expression */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAX 20

int top = -1;
char stack[MAX];
char pop();
void push(char item);

int prcd(char symbol)
{
   switch(symbol)
   {
      case '+':
      case '-':
         return 2;
         break;
      case '*':
      case '/':
         return 4;
         break;
      case '^':
      case '$':
         return 6;
         break;
      case '(':
      case ')':
      case '#':
         return 1;
         break;
   }
}

int isoperator(char symbol)
{
   switch(symbol)
   {
      case '+':
      case '-':
      case '*':
      case '/':
      case '^':
      case '$':
      case '(':
      case ')':
         return 1;
```

```
            break;
         default:
            return 0;
      }
   }

   void convertip(char infix[],char postfix[])
   {
      int i,symbol,j = 0;
      stack[++top] = '#';
      for(i=0;i<strlen(infix);i++)
      {
         symbol = infix[i];
         if(isoperator(symbol) == 0)
         {
            postfix[j] = symbol;
            j++;
         }
         else
         {
            if(symbol == '(')
               push(symbol);
            else if(symbol == ')')
            {
               while(stack[top] != '(')
               {
                  postfix[j] = pop();
                  j++;
               }
               pop(); //pop out (.
            }
            else
            {
               if(prcd(symbol) > prcd(stack[top]))
                  push(symbol);
               else
               {
                  while(prcd(symbol) <= prcd(stack[top]))
                  {
                       postfix[j] = pop();
                       j++;
                  }
                  push(symbol);
               }
            }
         }
      }

      while(stack[top] != '#')
      {
         postfix[j] = pop();
```

```
      j++;
   }
   postfix[j] = '\0';
}

main()
{
   char infix[20],postfix[20];
   clrscr();
   printf("Enter the valid infix string: ");
   gets(infix);
   convertip(infix, postfix);
   printf("The corresponding postfix string is: ");
   puts(postfix);
   getch();
}

void push(char item)
{
   top++;
   stack[top] = item;
}

char pop()
{
   char a;
   a = stack[top];
   top--;
   return a;
}
```

Output

```
Enter the valid infix string: (a+b*c)/(d$e)
The corresponding postfix string is: abc*+de$/

Enter the valid infix string: a*b+c*d/e
The corresponding postfix string is: ab*cd*e/+

Enter the valid infix string: a+b*c+(d*e+f)*g
The corresponding postfix string is: abc*+de*f+g*+
```

Result

Thus the given infix expression was converted into postfix form using stack.

Ex. No. 7b Postfix Expression Evaluation
Date:

Aim

To evaluate the given postfix expression using stack operations.

Algorithm

1. **Start**
2. **Define a array *stack* of size *max* = 20**
3. **Initialize *top* = -1**
4. **Read the postfix expression character-by-character**
   If character is an operand push it onto the stack
   If character is an operator
   Pop topmost two elements from stack.
   Apply operator on the elements and push the result onto the stack,
5. **Eventually only result will be in the stack at end of the expression.**
6. **Pop the result and print it.**
7. **Stop**

Program

```
/* Evaluation of Postfix expression using stack */

#include <stdio.h>
#include <conio.h>

struct stack
{
   int top;
   float a[50];
}s;

main()
{
   char pf[50];
   float d1,d2,d3;
   int i;
   clrscr();

   s.top = -1;
   printf("\n\n Enter the postfix expression: ");
   gets(pf);
   for(i=0; pf[i]!='\0'; i++)
   {
      switch(pf[i])
      {
         case '0':
         case '1':
         case '2':
         case '3':
         case '4':
         case '5':
         case '6':
         case '7':
         case '8':
         case '9':
            s.a[++s.top] = pf[i]-'0';
            break;

         case '+':
            d1 = s.a[s.top--];
            d2 = s.a[s.top--];
            s.a[++s.top] = d1 + d2;
            break;

         case '-':
            d2 = s.a[s.top--];
            d1 = s.a[s.top--];
            s.a[++s.top] = d1 - d2;
            break;
```

```
            case '*':
               d2 = s.a[s.top--];
               d1 = s.a[s.top--];
               s.a[++s.top] = d1*d2;
               break;

            case '/':
               d2 = s.a[s.top--];
               d1 = s.a[s.top--];
               s.a[++s.top] = d1 / d2;
               break;
         }
      }
      printf("\n Expression value is %5.2f", s.a[s.top]);
      getch();
   }
```

Output

```
Enter the postfix expression: 6523+8*+3+*
Expression value is 288.00
```

Result

Thus the given postfix expression was evaluated using stack.

Exp. No. 7c FCFS Scheduling
Date:

Aim

To schedule snapshot of processes queued according to FCFS scheduling.

Process Scheduling

- **CPU scheduling is used in multiprogrammed operating systems.**
- **By switching CPU among processes, efficiency of the system can be improved.**
- **Some scheduling algorithms are FCFS, SJF, Priority, Round-Robin, etc.**
- **Gantt chart provides a way of visualizing CPU scheduling and enables to understand better.**

First Come First Serve (FCFS)

- **Process that comes first is processed first**
- **FCFS scheduling is non-preemptive**
- **Not efficient as it results in long average waiting time.**
- **Can result in starvation, if processes at beginning of the queue have long bursts.**

Algorithm

1. **Define an array of structure *process* with members *pid, btime, wtime* & *ttime*.**
2. **Get length of the ready queue, i.e., number of process (say *n*)**
3. **Obtain *btime* for each process.**
4. **The *wtime* for first process is 0.**
5. **Compute *wtime* and *ttime* for each process as:**
   a. $wtime_{i+1} = wtime_i + btime_i$
   b. $ttime_i = wtime_i + btime_i$
6. **Compute average waiting time *awat* and average turnaround time *atur***
7. **Display the *btime, ttime* and *wtime* for each process.**
8. **Display GANTT chart for the above scheduling**
9. **Display *awat* time and *atur***
10. **Stop**

Program

```
/* FCFS Scheduling  - fcfs.c */

#include <stdio.h>

struct process
{
   int pid;
   int btime;
   int wtime;
   int ttime;
} p[10];

main()
{
   int i,j,k,n,ttur,twat;
   float awat,atur;

   printf("Enter no. of process : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
   {
      printf("Burst time for process P%d (in ms) : ",(i+1));
      scanf("%d", &p[i].btime);
      p[i].pid = i+1;
   }

   p[0].wtime = 0;
   for(i=0; i<n; i++)
   {
      p[i+1].wtime = p[i].wtime + p[i].btime;
      p[i].ttime = p[i].wtime + p[i].btime;
   }
   ttur = twat = 0;
   for(i=0; i<n; i++)
   {
      ttur += p[i].ttime;
      twat += p[i].wtime;
   }
   awat = (float)twat / n;
   atur = (float)ttur / n;

   printf("\n       FCFS Scheduling\n\n");
   for(i=0; i<28; i++)
      printf("-");
   printf("\nProcess B-Time T-Time W-Time\n");
   for(i=0; i<28; i++)
      printf("-");
```

```
    for(i=0; i<n; i++)
       printf("\n  P%d\t%4d\t%3d\t%2d",
                p[i].pid,p[i].btime,p[i].ttime,p[i].wtime);
    printf("\n");
    for(i=0; i<28; i++)
       printf("-");

    printf("\n\nAverage waiting time     : %5.2fms", awat);
    printf("\nAverage turn around time : %5.2fms\n", atur);

    printf("\n\nGANTT Chart\n");
    printf("-");
    for(i=0; i<(p[n-1].ttime + 2*n); i++)
       printf("-");
    printf("\n");
    printf("|");
    for(i=0; i<n; i++)
    {
       k = p[i].btime/2;
       for(j=0; j<k; j++)
          printf(" ");
       printf("P%d",p[i].pid);
       for(j=k+1; j<p[i].btime; j++)
          printf(" ");
       printf("|");
    }
    printf("\n");
    printf("-");
    for(i=0; i<(p[n-1].ttime + 2*n); i++)
       printf("-");
    printf("\n");
    printf("0");
    for(i=0; i<n; i++)
    {
       for(j=0; j<p[i].btime; j++)
          printf(" ");
       printf("%2d",p[i].ttime);
    }
}
```

Output

```
$ gcc fcfs.c

$ ./a.out
Enter no. of process : 4
Burst time for process P1 (in ms) : 10
Burst time for process P2 (in ms) : 4
Burst time for process P3 (in ms) : 11
Burst time for process P4 (in ms) : 6

       FCFS Scheduling

----------------------------
Process B-Time T-Time W-Time
----------------------------
  P1      10     10      0
  P2       4     14     10
  P3      11     25     14
  P4       6     31     25
----------------------------

Average waiting time     : 12.25ms
Average turn around time : 20.00ms


GANTT Chart
----------------------------------------
|     P1    |  P2 |     P3     |   P4  |
----------------------------------------
0           10    14           25      31
```

Result

Thus waiting time & turnaround time for processes based on FCFS scheduling was computed and the average waiting time was determined.

Ex. No. 8 Tree Traversal
Date:

Aim

To implement different types of traversal for the given binary search tree.

Algorithm

1. **Create a structure with key and 2 pointer variable left and right**
2. **Read the node to be inserted.**
   If (root==NULL)
   root=node
   else if (root->key < node->key)
   root->right=NULL
   else
   Root->left=node
3. **For Inorder Traversal**
   Traverse Left subtree
   Visit root
   Traverse Right subtree
4. **For Preorder Traversal**
   Visit root
   Traverse Left subtree
   Traverse Right subtree
5. **For Postorder Traversal**
   Traverse Left subtree
   Traverse Right subtree
   Visit root
6. **Stop.**

Program

```
/* Tree Traversal */

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

typedef struct node
{
   int data;
   struct node *left;
   struct node *right;
}node;

int count=1;

node *insert(node *tree,int digit)
{
   if(tree == NULL)
   {
      tree = (node *)malloc(sizeof(node));
      tree->left = tree->right=NULL;
      tree->data = digit;
      count++;
   }
   else if(count%2 == 0)
      tree->left = insert(tree->left, digit);
   else
      tree->right = insert(tree->right, digit);
   return tree;
}

void preorder(node *t)
{
   if(t != NULL)
   {
      printf(" %d", t->data);
      preorder(t->left);
      preorder(t->right);
   }
}

void postorder(node *t)
{
   if(t != NULL)
   {
      postorder(t->left);
      postorder(t->right);
      printf(" %d", t->data);
   }
}
```

```
void inorder(node *t)
{
   if(t != NULL)
   {
      inorder(t->left);
      printf(" %d", t->data);
      inorder(t->right);
   }
}

main()
{
   node *root = NULL;
   int digit;
   puts("Enter integer:To quit enter 0");
   scanf("%d", &digit);
   while(digit != 0)
   {
      root=insert(root,digit);
      scanf("%d",&digit);
   }
   printf("\nThe preorder traversal of tree is:\n");
   preorder(root);
   printf("\nThe inorder traversal of tree is:\n");
   inorder(root);
   printf("\nThe postorder traversal of tree is:\n");
   postorder(root);
   getch();
}
```

Output

```
Enter integer:To quit enter 0
12 4 6 9 14 17 3 19 0

The preorder traversal of tree is:
 12 4 9 17 19 6 14 3
The inorder traversal of tree is:
 19 17 9 4 12 6 14 3
The postorder traversal of tree is:
 19 17 9 4 3 14 6 12
```

Result

Thus three types of tree traversal was performed on the given binary tree.

Ex. No. 9 **Binary Search Tree**
Date:

Aim

To insert and delete nodes in a binary search tree.

Algorithm

1. **Create a structure with key and 2 pointer variable left and right.**
2. **Read the node to be inserted.**

```
If (root==NULL)
        root=node
else if (root->key<node->key)
        root->right=NULL
else
        Root->left=node
```

3. **For Deletion**

```
if it is a leaf node
        Remove immediately
        Remove pointer between del node & child
if it is having one child
        Remove link between del node&child
        Link delnode is child with delnodes parent
If it is a node with a children
        Find min value in right subtree
        Copy min value to delnode place
        Delete the duplicate
```

4. **Stop**

Program

```
/* Binary Search Tree */

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

struct node
{
   int key;
   struct node *left;
   struct node *right;
};

struct node *newNode(int item)
{
   struct node *temp = (struct node *)malloc(sizeof(struct
                                                        node));
   temp->key = item;
   temp->left = temp->right = NULL;
   return temp;
}

void inorder(struct node *root)
{
   if (root != NULL)
   {
      inorder(root->left);
      printf("%d ", root->key);
      inorder(root->right);
   }
}

struct node* insert(struct node* node, int key)
{
   if (node == NULL)
      return newNode(key);
   if (key < node->key)
      node->left  = insert(node->left, key);
    else
      node->right = insert(node->right, key);
   return node;
}

struct node * minValueNode(struct node* node)
{
   struct node* current = node;
   while (current->left != NULL)
      current = current->left;
   return current;
}
```

```
struct node* deleteNode(struct node* root, int key)
{
   struct node *temp;
   if (root == NULL)
      return root;
   if (key < root->key)
      root->left = deleteNode(root->left, key);
   else if (key > root->key)
      root->right = deleteNode(root->right, key);
   else
   {
      if (root->left == NULL)
      {
         temp = root->right;
         free(root);
         return temp;
      }
      else if (root->right == NULL)
      {
         temp = root->left;
         free(root);
         return temp;
      }
      temp = minValueNode(root->right);
      root->key = temp->key;
      root->right = deleteNode(root->right, temp->key);
   }
   return root;
}

main()
{
   struct node *root = NULL;
   root = insert(root, 50);
   root = insert(root, 30);
   root = insert(root, 20);
   root = insert(root, 40);
   root = insert(root, 70);
   root = insert(root, 60);
   root = insert(root, 80);
   printf("Inorder traversal of the given tree \n");
   inorder(root);
   printf("\nDelete 20\n");
   root = deleteNode(root, 20);
   printf("Inorder traversal of the modified tree \n");
   inorder(root);
   printf("\nDelete 30\n");
   root = deleteNode(root, 30);
   printf("Inorder traversal of the modified tree \n");
   inorder(root);
   printf("\nDelete 50\n");
```

```
   root = deleteNode(root, 50);
   printf("Inorder traversal of the modified tree \n");
   inorder(root);
}
```

Output

```
Inorder traversal of the given tree
20 30 40 50 60 70 80
Delete 20
Inorder traversal of the modified tree
30 40 50 60 70 80
Delete 30
Inorder traversal of the modified tree
40 50 60 70 80
Delete 50
Inorder traversal of the modified tree
40 60 70 80
```

Result

Thus nodes were inserted and deleted from a binary search tree.

Ex. No. 10a Linear Search
Date:

Aim

To perform linear search of an element on the given array.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i, *i = 0,1,2,…n–1***
4. **Read *search* value**
5. **Assign 0 to *found***
6. **Check each array element against *search***
   If A_i = *search* then
   ***found* = 1**
   Print "Element found"
   Print position *i*
   Stop
7. **If *found* = 0 then**
   print "Element not found"
8. **Stop**

Program

```
/* Linear search on a sorted array */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[50],i, n, val, found;
   clrscr();

   printf("Enter number of elements : ");
   scanf("%d", &n);
   printf("Enter Array Elements : \n");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   printf("Enter element to locate : ");
   scanf("%d", &val);
   found = 0;
   for(i=0; i<n; i++)
   {
      if (a[i] == val)
      {
         printf("Element found at position %d", i);
         found = 1;
         break;
      }
   }
   if (found == 0)
      printf("\n Element not found");
   getch();
}
```

Output

```
Enter number of elements : 7
Enter Array Elements :
23  6  12  5  0  32  10
Enter element to locate : 5
Element found at position 3
```

Result

Thus an array was linearly searched for an element's existence.

Ex. No. 10b Binary Search
Date:

Aim

To locate an element in a sorted array using Binary search method

Algorithm

1. **Start**
2. **Read number of array elements, say *n***
3. **Create an array *arr* consisting *n* sorted elements**
4. **Get element, say *key* to be located**
5. **Assign 0 to *lower* and *n* to *upper***
6. **While (*lower* < *upper*)**
   Determine middle element *mid* = (upper+lower)/2
   If key = arr[mid] then
   Print mid
   Stop
   Else if key > arr[mid] then
   lower = mid + 1
   else
   upper = mid – 1
7. **Print "Element not found"**
8. **Stop**

Program

```
/* Binary Search on a sorted array */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[50],i, n, upper, lower, mid, val, found;
   clrscr();

   printf("Enter array size  : ");
   scanf("%d", &n);
   for(i=0; i<n; i++)
      a[i] = 2 * i;

   printf("\n Elements in Sorted Order \n");
   for(i=0; i<n; i++)
      printf("%4d", a[i]);

   printf("\n Enter element to locate : ");
   scanf("%d", &val);
   upper = n;
   lower = 0;
   found = -1;
   while (lower <= upper)
   {
      mid = (upper + lower)/2;
      if (a[mid] == val)
      {
         printf("Located at position %d", mid);
         found = 1;
         break;
      }
      else if(a[mid] > val)
         upper = mid - 1;
      else
         lower = mid + 1;
   }

   if (found == -1)
      printf("Element not found");
   getch();
}
```

Output

```
Enter array size  : 9
 Elements in Sorted Order
   0   2   4   6   8  10  12  14  16
 Enter element to locate : 12
 Located at position 6



Enter array size  : 10
 Elements in Sorted Order
   0   2   4   6   8  10  12  14  16  18
 Enter element to locate : 13
 Element not found
```

Result

Thus an element is located quickly using binary search method.

Ex. No. 11a Bubble Sort
Date:

Aim

To sort an array of N numbers using Bubble sort.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i**
4. **Index *i* varies from 0 to *n*-2**
5. **Index *j* varies from *i*+1 to *n*-1**
6. **Traverse the array and compare each pair of elements**
   If $A_i > A_j$ then
   Swap A_i and *A*
7. **Stop**

Program

```
/* Bubble Sort */

#include <stdio.h>
#include <conio.h>

main()
{
   int a[50],i, j, n, t;
   clrscr();
   printf("Enter number of elements : ");
   scanf("%d", &n);
   printf("Enter Array Elements \n");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   for(i=0; i<n-1; i++)
   {
      for(j=i+1; j<n; j++)
      {
         if (a[i] > a[j])
         {
            t = a[i];
            a[i] = a[j];
            a[j] = t;
         }
      }
   }

   printf("\n Elements in Sorted order :");
   for(i=0; i<n; i++)
      printf("%d  ", a[i]);
   getch();
}
```

Output

```
Enter number of elements : 5
Enter Array Elements
3 7 -9 0 2

Elements in Sorted order :
-9  0  2  3  7
```

Result

Thus an array was sorted using bubble sort.

Ex. No. 11b Quick Sort
Date:

Aim

To sort an array of N numbers using Quick sort.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i**
4. **Select an pivot element *x* from A_i**
5. **Divide the array into 3 sequences: elements < *x*, *x*, elements > *x***
6. **Recursively quick sort both sets (A_i < *x* and *Ai* > *x*)**
7. **Stop**

Program

```
/* Quick Sort */

#include <stdio.h>
#include <conio.h>

void qsort(int arr[20], int fst, int last);

main()
{
   int arr[30];
   int i, size;
   printf("Enter total no. of the elements : ");
   scanf("%d", &size);
   printf("Enter total %d elements : \n", size);
   for(i=0; i<size; i++)
      scanf("%d", &arr[i]);
   qsort(arr,0,size-1);
   printf("\n Quick sorted elements \n");
   for(i=0; i<size; i++)
      printf("%d\t", arr[i]);
   getch();
}

void qsort(int arr[20], int fst, int last)
{
   int i, j, pivot, tmp;
   if(fst < last)
   {
      pivot = fst;
      i = fst;
      j = last;
      while(i < j)
      {
         while(arr[i] <=arr[pivot] && i<last)
         i++;
         while(arr[j] > arr[pivot])
         j--;
         if(i <j )
         {
            tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
         }
      }
      tmp = arr[pivot];
      arr[pivot] = arr[j];
      arr[j] = tmp;
      qsort(arr, fst, j-1);
      qsort(arr, j+1, last);
```

```
   }
}
```

Output

```
Enter total no. of the elements : 8
Enter total 8 elements :
1
2
7
-1
0
4
-2
3

Quick sorted elements
-2      -1      0       1       2       3       4       7
```

Result

Thus an array was sorted using quick sort's divide and conquer method.

Ex. No. 11c Merge Sort
Date:

Aim

To sort an array of N numbers using Merge sort.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i**
4. **Divide the array into sub-arrays with a set of elements**
5. **Recursively sort the sub-arrays**
6. **Merge the sorted sub-arrays onto a single sorted array.**
7. **Stop**

Program

```
/* Merge sort */

#include <stdio.h>
#include <conio.h>

void merge(int [],int ,int ,int );
void part(int [],int ,int );
int size;

main()
{
   int i, arr[30];
   printf("Enter total no. of elements : ");
   scanf("%d", &size);
   printf("Enter array elements : ");
   for(i=0; i<size; i++)
      scanf("%d", &arr[i]);
   part(arr, 0, size-1);
   printf("\n Merge sorted list : ");
   for(i=0; i<size; i++)
      printf("%d ",arr[i]);
   getch();
}

void part(int arr[], int min, int max)
{
   int mid;
   if(min < max)
   {
      mid = (min + max) / 2;
      part(arr, min, mid);
      part(arr, mid+1, max);
      merge(arr, min, mid, max);
   }
   if (max-min == (size/2)-1)
   {
      printf("\n Half sorted list : ");
      for(i=min; i<=max; i++)
         printf("%d ", arr[i]);
   }
}

void merge(int arr[],int min,int mid,int max)
{
   int tmp[30];
   int i, j, k, m;
   j = min;
   m = mid + 1;
```

```
   for(i=min; j<=mid && m<=max; i++)
   {
      if(arr[j] <= arr[m])
      {
         tmp[i] = arr[j];
         j++;
      }
      else
      {
         tmp[i] = arr[m];
         m++;
      }
   }
   if(j > mid)
   {
      for(k=m; k<=max; k++)
      {
         tmp[i] = arr[k];
         i++;
      }
   }
   else
   {
      for(k=j; k<=mid; k++)
      {
         tmp[i] = arr[k];
         i++;
      }
   }
   for(k=min; k<=max; k++)
      arr[k] = tmp[k];
}
```

Output

```
Enter total no. of elements : 8
Enter array elements : 24 13 26 1 2 27 38 15

 Half sorted list : 1 13 24 26
 Half sorted list : 2 15 27 38
 Merge sorted list : 1 2 13 15 24 26 27 38
```

Result

Thus array elements was sorted using merge sort's divide and conquer method.

Ex. No. 11d Insertion Sort
Date:

Aim

To sort an array of N numbers using Insertion sort.

Algorithm

1. **Start**
2. **Read number of array elements *n***
3. **Read array elements A_i**
4. **Sort the elements using insertion sort**
   In pass p, move the element in position p left until its correct place is found among the first p + 1 elements.
   Element at position p is saved in temp, and all larger elements (prior to position p) are moved one spot to the right. Then temp is placed in the correct spot.
5. **Stop**

Program

```
/* Insertion Sort */

main()
{
   int i, j, k, n, temp, a[20], p=0;

   printf("Enter total elements: ");
   scanf("%d",&n);
   printf("Enter array elements: ");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);

   for(i=1; i<n; i++)
   {
      temp = a[i];
      j = i - 1;
      while((temp<a[j]) && (j>=0))
      {
         a[j+1] = a[j];
         j = j - 1;
      }
      a[j+1] = temp;

      p++;
      printf("\n After Pass %d: ", p);
      for(k=0; k<n; k++)
         printf("  %d", a[k]);
   }

   printf("\n Sorted List : ");
   for(i=0; i<n; i++)
      printf(" %d", a[i]);
}
```

Output

```
Enter total elements: 6
Enter array elements: 34 8 64 51 32 21
 After Pass 1:   8  34  64  51  32  21
 After Pass 2:   8  34  64  51  32  21
 After Pass 3:   8  34  51  64  32  21
 After Pass 4:   8  32  34  51  64  21
 After Pass 5:   8  21  32  34  51  64
 Sorted List :  8 21 32 34 51 64
```

Result

Thus array elements was sorted using insertion sort.

Ex. No. 12 Open Addressing Hashing Technique
Date:

Aim

To implement hash table using a C program.

Algorithm

1. **Create a structure, data (hash table item) with key and value as data.**
2. **Now create an array of structure, data of some certain size (10, in this case). But, the size of array must be immediately updated to a prime number just greater than initial array capacity (i.e 10, in this case).**
3. **A menu is displayed on the screen.**
4. **User must choose one option from four choices given in the menu**
5. **Perform all the operations**
6. **Stop**

Program

```
/* Open hashing */

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

#define MAX 10

main()
{
   int a[MAX], num, key, i;
   char ans;
   int create(int);
   void linearprobing(int[], int, int);
   void display(int[]);

   printf("\nCollision handling by linear probing\n\n");
   for(i=0; i<MAX; i++)
      a[i] = -1;
   do
   {
      printf("\n Enter number:");
      scanf("%d", &num);
      key = create(num);
      linearprobing(a, key, num);
      printf("\nwish to continue?(y/n):");
      ans = getch();
   } while( ans == 'y');
   display(a);
}

int create(int num)
{
   int key;
   key = num % 10;
   return key;
}

void linearprobing(int a[MAX], int key, int num)
{
   int flag, i, count = 0;
   void display(int a[]);
   flag = 0;
   if(a[key] == -1)
      a[key] = num;
   else
   {
      i=0;
```

```
      while(i < MAX)
      {
         if(a[i] != -1)
         count++;
         i++;
      }
      if(count == MAX)
      {
         printf("hash table is full");
         display(a);
         getch();
         exit(1);
      }
      for(i=key+1; i<MAX; i++)
         if(a[i] == -1)
         {
            a[i] = num;
            flag = 1;
            break;
         }
      for(i=0; i<key && flag==0; i++ )
      if(a[i] == -1)
      {
         a[i] = num;
         flag = 1;
         break;
      }
   }
}

void display(int a[MAX])
{
   int i;
   printf("\n Hash table is:");
   for(i=0; i<MAX; i++)
      printf("\n %d\t\t%d",i,a[i]);
}
```

Output

```
Collision handling by linear probing
 Enter number:1
wish to continue?(y/n):
 Enter number:26
wish to continue?(y/n):
 Enter number:62
wish to continue?(y/n):
 Enter number:93
wish to continue?(y/n):
```

```
 Enter number:84
wish to continue?(y/n):
 Enter number:15
wish to continue?(y/n):
  Enter number:76
wish to continue?(y/n):
 Enter number:98
wish to continue?(y/n):
 Enter number:26
wish to continue?(y/n):
 Enter number:199
wish to continue?(y/n):
 Enter number:1234
wish to continue?(y/n):
 Enter number:5678
hash table is full
 Hash table is:
 0              1234
 1              1
 2              62
 3              93
 4              84
 5              15
 6              26
 7              76
 8              98
 9              199
```

Result

Thus hashing has been performed successfully.