**Program 1. Write a menu driven program to implement linear and binary search also find the location of its first occurrence.**

**Algorithm :**

Step 1: Start

Step 2: Array element and the required variables are declared

a[10], i , n, item, pos, j, mid, low, high, temp]

**functions are** : lsearch(int a[],int item,int n)

bsearch(int a[], int item,int n)

Step 3: Accept the array elements

For i=0 to n by 1

Read : a[i]

Step 4: Accept the item to be searched

**Function definition:lsearch**

Step 1: for i=0 to n by 1

If a[i] = = item

pos 🡨 i+1

[end of if]

[end of for]

Step 2: if pos! =-1

Write “item found and position ”

else

write “item not found”

**Function definition:bsearch**

Step 1: [Initialize low:=0 ,high = n-1]

Step 2: for i=0 to n-1 by 1

if ( a[i] > a[j] )

temp 🡨 a[i]

a[i] 🡨 a[j]

Step 3: Display the array elements

Step 4: While (low<=high) do

mid🡨 ( low + high ) / 2

if (a[mid] = = item)

pos🡨mid+1

[end if]

else if ( a [mid] >item)

high🡨mid-1

else

low🡨mid+1

[End of while]

Step 5: if pos! =-1

Write “item found and position”

else

Write “item not found”

**Main function:**

Accept the choice from the user and call the functions in order to perform the operations according to users choice.

**Program:**

//Q.1: Write a menu driven program to implement linear and binary search also find the

//location of its first occurrence

#include <stdio.h>

#include <stdlib.h>

// Set of useful variables in stracture

struct st{

int n,arr[100],key;

};

//Input data from keyboard and pass to main()

struct st getData(){

struct st X;

int i;

printf("How many numbers? ");

scanf("%d", &X.n);

printf("Enter integrs one by one: ");

for(i = 0; i<X.n; i++){

scanf("%d", &X.arr[i]);

}

printf("What you want to search? ");

scanf("%d", &X.key);

return X;

}

//Execute Linear Search

void linear\_Search(int key, int n, int arr[])

{

int i,flag = 0;

for(i = 0; i <n; i++){

if(key == arr[i]){

flag = 1;

printf("%d found at position %d.\n",key,i+1);

}

}

if(flag == 0)

printf("%d not found in the list",key);

}

//Execute Binary Search

void binary\_Search(int key, int n, int arr[])

{

int i,j,temp;

int low,high,mid,location;

int flag = 0;

for(i = 0; i<=n; ++i){

for(j = i+1; j<=n; ++j){

if(arr[i] > arr[j]){

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

low = 0;

high = n-1;

while(low<=high){

mid = (low + high) / 2;

if(arr[mid] == key && n%2 == 0){

location = mid;

flag = 2;

break;

}

else if(arr[mid] == key && n%2 != 0){

location = mid;

flag = 1;

break;

}

else if(arr[mid] > key)

high = mid - 1;

else if(arr[mid] < key)

low = mid + 1;

}

if(flag == 1)

printf("%d found at location %d.\n", key, location);

else if(flag == 2)

printf("%d found at location %d.\n", key, location+1);

else

printf("%d Not found! in the list.\n", key);

}

int main() {

struct st X;

int ch;

//Select Choice

while(1){

printf("\n\n\*\*\*SEARCHING\*\*\*\n\n");

printf("Linear Search: [PRESS 1]\n");

printf("Binary Search: [PRESS 2]\n");

printf("EXIT: [PRESS 0]\n");

scanf("%d", &ch);

switch(ch){

case 1:

X = getData();

linear\_Search(X.key,X.n,X.arr);

break;

case 2:

X = getData();

binary\_Search(X.key,X.n,X.arr);

break;

case 0:

exit(1);

return 0;

default:

printf("\nWrong Choice !\n");

}

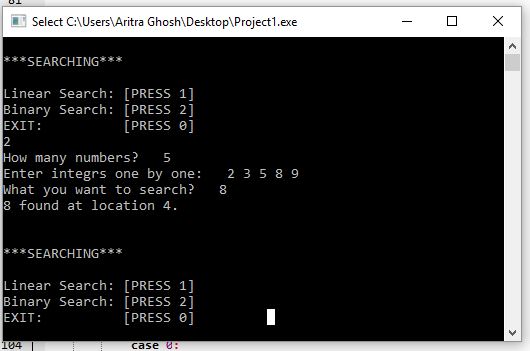
}

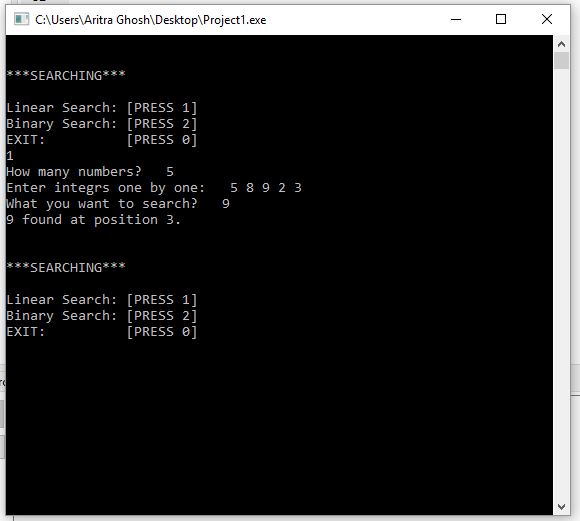
return 0;

system("PAUSE");

}

**Output:**

****

****

**Program 2: A menu driven program to sort the array in ascending /descending**

**order using a) Quick sort b) merge sort.**

**Algorithm :**

Step 1: Declare the size of the array elements and declare the functions within

the functions and define them and call them respectively

**functions are:**

mergesort (int a[],int lb,int ub)

merge (int a[],int mid,int lb,int ub)

quicksort(int a[],int lb,int ub)

Step 2 : Read the array elements

For i=0 to n by 1

Read: a[i]

**function definition :quicksort**

Step 1: declare the variables up,down,temp,key ,flag=1

Step 2: if(lb<ub)

Step 3: Up🡨lb

down🡨ub

key🡨a[lb]

Step 4: while(flag=1) do

up🡨up+1

Step 5: while(a[up]<key) do

up🡨up+1

[end of while]

Step 6: while(a[down]>key) do

Down🡨down-1

Step7: If(up<down) then

temp🡨a[up]

a[up] 🡨a[down]

\ a[down] 🡨temp

[end of if]

else

flag=0

[end of while]

Step 8: temp🡨a[lb]

Step 9: a[lb] 🡨a[down]

Step 10: a[down] 🡨temp

Step 11: **Call function** quicksort(a,lb,down-1)

Step 12: **Call function** quicksort(a,down+1,ub)

**Function definition : mergesort**

Step 1: declare mid

Step 2: if(lb<ub)

mid🡨(lb+ub)/2

**call functions:**

mergesort(a,lb,mid)

mergesort(a,mid+1,ub)

merge(a,lb,mid,ub)

[end of if ]

**Function definition : merge**

Step 1: An array and the required variables are declared and initialized

[ j, k, c[20] , i 🡨 lb k 🡨 lb, j🡨mid+1 ]

Step 2 : while((i<=mid) && (j<=high)) do

if(a[i]<a[j])

c[k]🡨a[i]

k🡨k+1

i🡨i+1

[end of if]

else

c[k]🡨a[j]

k🡨k+1

j🡨j+1

[end of else]

[end of while]

Step 3: while(i<=mid) do

c[k]🡨a[i]

k🡨k+1

i🡨i+1

[end of while]

Step 4: while(j<=ub) do

c[k]🡨a[j]

k🡨k+1

j🡨j+1

[end of while ]

Step 5: For : i=lb to i<=k-1 by 1

a[i]🡨c[i]

**Main function :**

Step 1: accept the choice from the user and call the

functions in order to perform the operations according to users choice.

**Program:**

//Q.2: Write a menu driven program to sort the array in ascending/descending order using

//a) Quick sort b) Merge sort

#include <stdio.h>

#include <stdlib.h>

// A utility function to swap two elements

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int arr[], int low, int high)

{

int j;

int pivot = arr[high]; // pivot

int i = (low - 1); // Index of smaller element

for(j = low; j <= high- 1; j++)

{

// If current element is smaller than or

// equal to pivot

if (arr[j] <= pivot)

{

i++; // increment index of smaller element

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now

at right place \*/

int pi = partition(arr, low, high);

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l+(r-l)/2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

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

printf("%d ", arr[i]);

printf("\n");

}

int main()

{

int n,i;

char ch;

int arr[100];

printf("How many Numbers? ");

scanf("%d", &n);

for(i = 0; i<n; i++){

scanf("%d", &arr[i]);

}

printf("\n\n");

while(1){

printf("\n\nQuick sort [PRESS 'A']\nMerge sort [PRESS 'B']\nExit [PRESS 0]\n\n");

scanf("%c",&ch);

switch(ch){

case 'A':

quickSort(arr, 0, n-1);

printf("Sorted array: \n");

printArray(arr, n);

break;

case 'B':

mergeSort(arr, 0, n - 1);

printf("\nSorted array is \n");

printArray(arr, n);

break;

case '0':

printf("\nThans for Exit!\n");

exit(1);

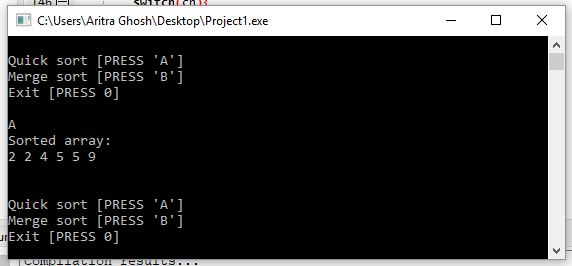
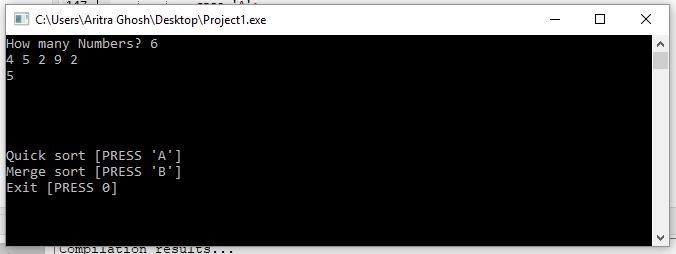
}

}

return 0;

}

**Output:**



**Program 3: A menu driven program to create a linked list and to perform insert and delete operations.**

**Algorithm:**

Step 1: Start

Step 2: The data and link part of the node is being declared within the structure

and initially a pointer ‘h’ of node type which belongs to structure is being

initialized to NULL with the help of constructor in the class ‘list’ and

functions are being declared in the class and defined outside the class

separately which are used in main function. The display function is used to

display the elements in the singly linked list.

The function used are as follows:

**Functions:**

void create( )

void insert( )

void disp( )

int count( )

void del( )

**Creation of node**

Step 1: create a pointer ‘h’ to point to the structure called node

Step 2 : create a node dynamically i.e. allocate memory for storing this

Structure Using malloc function and assign to ‘t’

t🡨(struct node \*) malloc (sizeof(struct node))

Step 3: When the node is being created information and link part should

be given data

Step 4: When the link part of the node points to NULL, it indicates that the node

Does not point to anything.

**Creation of linked list**

Step 1: Initially ‘h’ pointer points to NULL, indicating node is empty

Step 2: Another pointer ‘p’ points to first node i.e ‘h’

[Initialize p:=h]

Step 3: create a new node pointed by ‘p’ pointer

Step 4: Read in the data element and store the data field

t->data=num

t->link=NULL

Step 5: If (p = = NULL), then this new node is first node

h🡨t

p🡨h

else

p = p --🡪 link

p🡪link = t

**Insertion**

Step 1: declare 2 variables I, pos, num

Step 2: declare 2 pointers p,t of node type

Step 3: count the no. of nodes

**call the function count( )** and assign to i

Step 4 :Accept the position

Step 5 : create a new node assign address to t

Step 6 : Enter the element to data field

t->data=num

t->link=NULL

[Initialize p=h]

i.e p points to first node

Step 7: If item to be inserted at first position

Then

h=t

t->link=p

else

p = p -🡪link

p🡪link = t

Step 8 : while(i<pos-1) do Steps 9,10

step 9: p = p 🡪link

step 10: i = i+1

Step 11: [Make connection between new node and next node]

t🡪link = p🡪link

Step 12: p🡪link = t

[make connection between current pointer and new node]

[End while ]

**Deletion**

Step 1: declare variables I, pos and two pointers p,prev

Step 2 : Call the function count and assign to i

I = count( )

Step 3: accept position

[Initialize p:=h]

Step 4: if pos = = 1 then

h = p 🡪 link

else

Step 5: For i= 1 to i < pos by 1

prev = p

p = p 🡪 link

[ end of for loop ]

[ end of else ]

Step 6: Prev 🡪 link = p 🡪 link

Free ( p )

[ delete the node pointed by ‘p’ pointer ]

**Counting the no. of nodes**

Step 1: declare and initialize pointer ‘p ’ to first node

[ p = h ]

Step 2: initialize count = 0

Step 3 : if p = = NULL

Write “ list is empty “

Else

Step 4 : While ( p!=NULL) do Step 5

Step 5: Count 🡨 count + 1

p= p 🡪link

Return count

[ end of else ]

**Main function:**

accept the choice from the user and call the functions in order to perform

the operations according to users choice.

**Program:**

//03. Write a menu driven program to create a linked list and to perform insert and delete

//operations

#include<stdio.h>

#include<stdlib.h>

/\*----Function Prototypes-----\*/

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

/\*-----------------------------\*/

struct node

{

int info;

struct node \*next;

};

struct node \*start=NULL;

int main() //main() starts

{

int choice;

while(1){

printf("\n\*\*\*SINGLE LINKED LIST OPERATIONS:\*\*\*\*\n");

printf("\n MENU \n");

printf("---------------------------------------\n");

printf("\n 1.Create \n");

printf("\n 2.Display \n");

printf("\n 3.Insert at the beginning \n");

printf("\n 4.Insert at the end \n");

printf("\n 5.Insert at specified position \n");

printf("\n 6.Delete from beginning \n");

printf("\n 7.Delete from the end \n");

printf("\n 8.Delete from specified position \n");

printf("\n 9.Exit \n");

printf("\n--------------------------------------\n");

printf("Enter your choice:\t");

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

display();

break;

case 2:

display();

break;

case 3:

insert\_begin();

break;

case 4:

insert\_end();

break;

case 5:

insert\_pos();

break;

case 6:

delete\_begin();

break;

case 7:

delete\_end();

break;

case 8:

delete\_pos();

break;

case 9:

exit(0);

break;

default:

printf("\n Wrong Choice:\n");

break;

}//end of switch()

}

return 0;

}//end of main()

void create()

{

struct node \*temp,\*ptr;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nOut of Memory Space:\n");

exit(0);

}

printf("\nEnter the data value for the node:\t");

scanf("%d",&temp->info);

temp->next=NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

printf("Linked List Created.\n");

}//end of create()

void display()

{

struct node \*ptr;

if(start==NULL)

{

printf("\nList is empty:\n");

return;

}

else

{

ptr=start;

printf("\nThe List elements are:\n");

while(ptr!=NULL)

{

printf("%d\t",ptr->info );

ptr=ptr->next ;

}//end of while

}//end of else

printf("\n\n");

}//end of display()

void insert\_begin()

{

struct node \*temp;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nOut of Memory Space:\n");

return;

}

printf("\nEnter the data value for the node:\t" );

scanf("%d",&temp->info);

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

temp->next=start;

start=temp;

}

}//end of insert\_begin()

void insert\_end()

{

struct node \*temp,\*ptr;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nOut of Memory Space:\n");

return;

}

printf("\nEnter the data value for the node:\t" );

scanf("%d",&temp->info );

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next !=NULL)

{

ptr=ptr->next ;

}

ptr->next =temp;

}

}//end of insert\_end

void insert\_pos()

{

struct node \*ptr,\*temp;

int i,pos;

temp=(struct node \*)malloc(sizeof(struct node));

if(temp==NULL)

{

printf("\nOut of Memory Space:\n");

return;

}

printf("\nEnter the position for the new node to be inserted:\t");

scanf("%d",&pos);

printf("\nEnter the data value of the node:\t");

scanf("%d",&temp->info) ;

temp->next=NULL;

if(pos==0)

{

temp->next=start;

start=temp;

}

else

{

for(i=0,ptr=start;i<pos-1;i++)

{

ptr=ptr->next;

if(ptr==NULL)

{

printf("\nPosition not found:[Handle with care]\n");

return;

}

}

temp->next =ptr->next ;

ptr->next=temp;

}//end of else

}//end of insert\_pos

void delete\_begin()

{

struct node \*ptr;

if(ptr==NULL)

{

printf("\nList is Empty:\n");

return;

}

else

{

ptr=start;

start=start->next ;

printf("\nThe deleted element is :%d\t",ptr->info);

free(ptr);

}

}//end of delete\_begin()

void delete\_end()

{

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("\nList is Empty:");

exit(0);

}

else if(start->next ==NULL)

{

ptr=start;

start=NULL;

printf("\nThe deleted element is:%d\t",ptr->info);

free(ptr);

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

temp=ptr;

ptr=ptr->next;

}

temp->next=NULL;

printf("\nThe deleted element is:%d\t",ptr->info);

free(ptr);

}

}//end of delete\_begin()

void delete\_pos()

{

int i,pos;

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("\nThe List is Empty:\n");

exit(0);

}

else

{

printf("\nEnter the position of the node to be deleted:\t");

scanf("%d",&pos);

if(pos==0)

{

ptr=start;

start=start->next ;

printf("\nThe deleted element is:%d\t",ptr->info );

free(ptr);

}

else

{

ptr=start;

for(i=0;i<pos;i++)

{

temp=ptr;

ptr=ptr->next ;

if(ptr==NULL)

{

printf("\nPosition not Found:\n");

return;

}

}

temp->next =ptr->next ;

printf("\nThe deleted element is:%d\t",ptr->info );

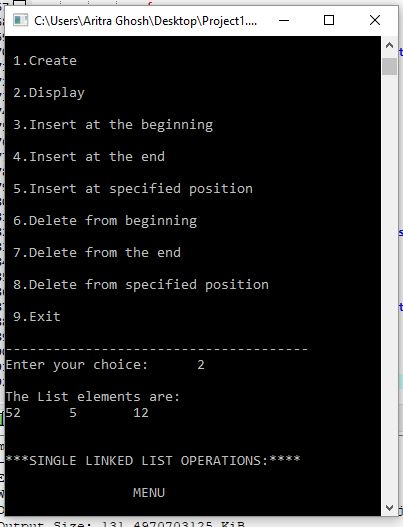
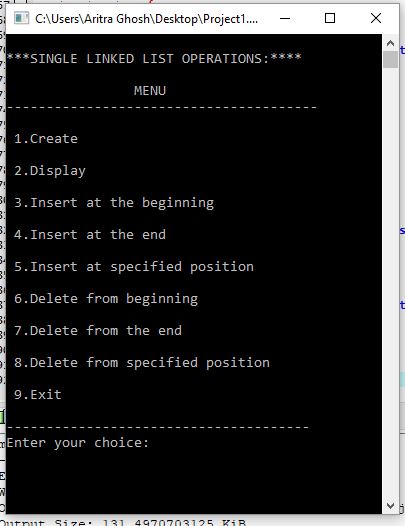
free(ptr);

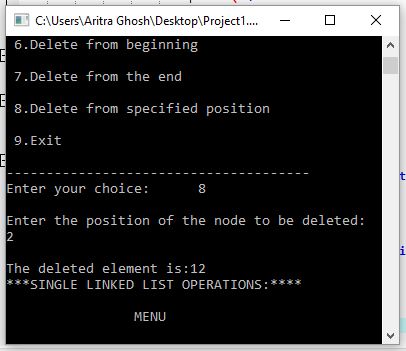
}

}//end of else

}//end of delete\_pos()

**Output:**

****



**Program 4: A program to add two polynomials using a linked list.**

**Algorithm :**

Step 1: Declare a structure\_ element with the following parameters

Coeff ,degree of type int, next of type element

Step 2: Declare a structure \_poly with degree of type int and first of type element

**Functions used:**

( Create a new polynomial)

poly \* new\_poly  ()  
(  Populate a new polynomial)  
void   get\_poly  (poly \*p)   
( Display a given polynomial)  
void   disp\_poly (poly \*p)  
( Create a new polynomial as the sum of two given polynomials)  
poly \* add\_poly (poly \*p1, poly \*p2)

**Main Function**

Step 1: create two pointers p1,p2 of type poly

Step 2: p1 🡨new\_poly()  
  **call function get\_poly(p1)  
  call function disp\_poly(p1**)

Step 3 : p2🡨new\_poly()  
 **call function get\_poly(p1)  
  call function disp\_poly(p1)**

Step 4: disp\_poly(add\_poly(p1,p2)) ( this will display the sum of the two

polynomials)

**Functions :**

**1**.**poly \* new\_poly ()**

Step1:  poly \*p  
  Step 2:p 🡨 (poly \*) malloc (sizeof (poly))  
 Step 3:p->degree 🡨0  
  Step 4: p->first🡨NULL  
   Step 5: return p

**2**.**void get\_poly (poly \*p)**

Step 1: declare a, i, coef as integers  
  Step 2: Declare \*next,\*current of type element

Step 3: Read the order of the first polynomial

  Step 4: p->degree 🡨a  
  Step 5: next 🡨NULL  
 Step 6:  for i🡨0 to p->degree by 1   
 **Read the coefficient**

    current 🡨 (element \*) malloc (sizeof(element))  
    current->degree 🡨i     
    current->coef  🡨coef  
    current->next  🡨NULL  
    if (next != NULL)

current->next 🡨next  
    next 🡨current

  [end of if]  
   p->first 🡨next

[end of for]

**3**. **void disp\_poly (poly \*p)**

  Step 1: Declare member as a pointer of type element  
  Step 2: Write the order of the polynomial as p->degree  
   Step 3:member 🡨p->first  
  Step 4:while member != NULL do   
    if (member -> next != NULL)  
      write member->coef, member->degree

[end of if]  
    else  
 write member->coef, member->degree  
    member = member -> next

[end of while]  
   
  **4. poly \* add\_poly (poly \*p1, poly \*p2)**

Step 1: declare \*higher, \*lower, \*result of type poly

Step 2: declare \*member1, \*member2, member3 of type element  
 Step 3: if (p1->degree > p2->degree)   
    higher 🡨p1  
    lower  🡨p2  
  [end of if]   
  else

   higher 🡨p2  
    lower  🡨p1

[end of else]  
  Step 4:result 🡨higher

Step 5:member1 🡨result -> first

Step 6:member2 🡨lower -> first

Step 7:while (member1 != NULL)

if (member1->degree == member2->degree)

member1->coef 🡨member2->coef+ member1->coef

member2 🡨member2->next

[end of if]

member1 🡨member1->next

[end of while]

Step 8:return(result)

**Program :**

#include<stdio.h>

#include<stdlib.h>

typedef struct link {

    int coeff;

    int pow;

    struct link \* next;

} my\_poly;

/\*\* The prototypes \*/

void my\_create\_poly(my\_poly \*\*);

void my\_show\_poly(my\_poly \*);

void my\_add\_poly(my\_poly \*\*, my\_poly \*, my\_poly \*);

/\*\*

 \* The simple menu driven main function

 \*/

int main(void) {

    int ch;

    do {

        my\_poly \* poly1, \* poly2, \* poly3;

        printf("\nCreate 1st expression\n");

        my\_create\_poly(&poly1);

        printf("\nStored the 1st expression");

        my\_show\_poly(poly1);

        printf("\nCreate 2nd expression\n");

        my\_create\_poly(&poly2);

        printf("\nStored the 2nd expression");

        my\_show\_poly(poly2);

        my\_add\_poly(&poly3, poly1, poly2);

        my\_show\_poly(poly3);

        printf("\nAdd two more expressions? (Y = 1/N = 0): ");

        scanf("%d", &ch);

    } while (ch);

    return 0;

}

void my\_create\_poly(my\_poly \*\* node) {

    int flag; //A flag to control the menu

    int coeff, pow;

    my\_poly \* tmp\_node; //To hold the temporary last address

    tmp\_node = (my\_poly \*) malloc(sizeof(my\_poly)); //create the first node

    \*node = tmp\_node; //Store the head address to the reference variable

    do {

        //Get the user data

        printf("\nEnter Coeff:");

        scanf("%d", &coeff);

        tmp\_node->coeff = coeff;

        printf("\nEnter Pow:");

        scanf("%d", &pow);

        tmp\_node->pow = pow;

        //Done storing user data

        //Now increase the Linked on user condition

        tmp\_node->next = NULL;

        //Ask user for continuation

        printf("\nContinue adding more terms to the polynomial list?(Y = 1/N = 0): ");

        scanf("%d", &flag);

        //printf("\nFLAG: %c\n", flag);

        //Grow the linked list on condition

        if(flag) {

            tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly)); //Grow the list

            tmp\_node = tmp\_node->next;

            tmp\_node->next = NULL;

        }

    } while (flag);

}

/\*\*

 \* The show polynomial function

 \* Prints the Polynomial in user readable format

 \* @param my\_poly \* node The polynomial linked list

 \* @return void

 \*/

void my\_show\_poly(my\_poly \* node) {

    printf("\nThe polynomial expression is:\n");

    while(node != NULL) {

        printf("%dx^%d", node->coeff, node->pow);

        node = node->next;

        if(node != NULL)

            printf(" + ");

    }

}

/\*\*

 \* The polynomial add function

 \* Adds two polynomial to a given variable

 \* @param my\_poly \*\* result Stores the result

 \* @param my\_poly \* poly1 The first polynomial expression

 \* @param my\_poly \* poly2 The second polynomial expression

 \* @return void

 \*/

void my\_add\_poly(my\_poly \*\* result, my\_poly \* poly1, my\_poly \* poly2) {

    my\_poly \* tmp\_node; //Temporary storage for the linked list

    tmp\_node = (my\_poly \*) malloc(sizeof(my\_poly));

    tmp\_node->next = NULL;

    \*result = tmp\_node; //Copy the head address to the result linked list

    //Loop while both of the linked lists have value

    while(poly1 && poly2) {

        if (poly1->pow > poly2->pow) {

            tmp\_node->pow = poly1->pow;

            tmp\_node->coeff = poly1->coeff;

            poly1 = poly1->next;

        }

        else if (poly1->pow < poly2->pow) {

            tmp\_node->pow = poly2->pow;

            tmp\_node->coeff = poly2->coeff;

            poly2 = poly2->next;

        }

        else {

            tmp\_node->pow = poly1->pow;

            tmp\_node->coeff = poly1->coeff + poly2->coeff;

            poly1 = poly1->next;

            poly2 = poly2->next;

        }

        //Grow the linked list on condition

        if(poly1 && poly2) {

            tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly));

            tmp\_node = tmp\_node->next;

            tmp\_node->next = NULL;

        }

    }

    //Loop while either of the linked lists has value

    while(poly1 || poly2) {

        //We have to create the list at beginning

        //As the last while loop will not create any unnecessary node

        tmp\_node->next = (my\_poly \*) malloc(sizeof(my\_poly));

        tmp\_node = tmp\_node->next;

        tmp\_node->next = NULL;

        if(poly1) {

            tmp\_node->pow = poly1->pow;

            tmp\_node->coeff = poly1->coeff;

            poly1 = poly1->next;

        }

        if(poly2) {

            tmp\_node->pow = poly2->pow;

            tmp\_node->coeff = poly2->coeff;

            poly2 = poly2->next;

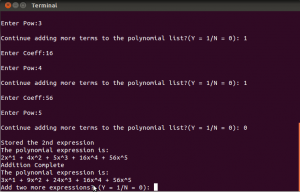
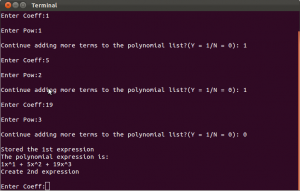
        }

    }

    printf("\nAddition Complete");

}

**Output:**



**Program 5: A menu driven program to perform insert and delete operations in a circular linked list.**

**Algorithm:**

Step 1: The data and link part of the node is being declared within the structure and initially a pointer ‘h’ of node type which belongs to structure is being initialized to NULL with the help of constructor in the class clist and functions are being declared in the class and defined outside the class separately which are used in main function. The display function is used to display the elements in the circular list. The functions used are as follows:

**Functions:**

void create( )

void insert( )

void disp( )

int count( )

void del( )

**Creation of node**

Step 1: Declaration two pointers‘t’, ‘p’ of node type, three variables i, n, num and

initialization of first node i.e. head node ‘h’ to NULL and pointer p stores the address of the head node

h🡨NULL

p 🡨 h

Step 2: accept the number of elements to be added

Step 3: for I = 1 to i<=n by 1

[accept the data element to be added within the loop ]

allocate the memory for the node to be created using malloc function and

the pointer t holds the address of the memory and input the element in the

data part of the node which is being created by the ‘t’ pointer.

t = (struct node\*)malloc(sizeof(struct node))

t->data=num

Step 4: The link part of the first node is being pointed to NULL

if(p = = NULL) then

h = t

t->link = h

p = h

[end of if]

else

while(p->link != h) do Step 5

step 5 : p = p->link

t->link = h

p->link = t

[end of else]

[end of for loop ]

**Insertion into the circular linked list**

Step 1: declare 2 variables I, pos, num

Step 2: declare 2 pointers p, t, q of node type

Step 3: count the no. of nodes

Call the function count and assign to i

I = count ( )

Step 3: Accept the position and check for valid position

[allocate the memory for node pointed by ‘t’ pointer using malloc function so pointer ‘t’ holds the address of the node]

t🡨(struct node \*) malloc (sizeof(struct node))

Step 4: Enter the element to data field

t->data=num

[Initialize p:=q= h]

i.e both p and q holds the address of first or head node ]

Step 5: if (pos == 1) then

[address of data field is stored in first node]

h = t;

if (p != NULL) then

[address of pointer ‘p’ is stored in link part of the node]

t->link = p

else

[ hold the address of head node in link part]

t->link = h

[both ‘q’ and ‘p’ pointer addresses are same]

p = q;

if (p != NULL) then

while(p->link != q) do step 6

Step 6:

[ store the address of next node link part into ‘ p ‘ pointer and address ]

Head node is stored in link]

p=p->link;

p->link = h;

[end of if ]

[end of outer if structure ]

else

if(pos = = i+1) then

while(p->link!=h) do step 7

Step 7: [store the address of the link field of the node into p pointer,

link part address of ‘t’ pointer into link and address head

node is stored in link part of ‘t’ pointer]

p=p->link

p->link=t t->link=h

[end of if structure ]

else

for i=1 to i<pos-1 by 1

p = p->link;

[make connection between new node and next node and current pointer and new node ]

t->link=p->link

p->link=t

[end of inner and outer else structure ]

**Deletion of node**

Step 1: Declare variables I, pos and three pointers p, prev, q of node type

Step 2: Call the function count and assign to i

I = count ( )

If (I = = 0) then

Write “list is empty”

Step 3: Accept position and check for valid position i.e.

[Initialize p = q =h i.e both p and q pointers stores the address

head node]

if (pos = = 1 and count ( ) = = 1) then

[ head node pointer points to NULL ]

H = NULL

Delete the ‘p’ pointer which has the current node address

[end of if]

Step4: If (pos == 1) then

Address of link is stored in head node pointer

h= p->link

While (p->link!=q) do step 5

Step 5: the link part address is stored in ‘p’ pointer and now the

head node Address is stored in link, now delete the

node pointed by ‘q’ pointer

p=p->link

p->link =h

free (q)

[end of if ]

Step 6: Within the for loop the address of node pointed by ‘p’ pointer is

stored into prev pointer hich points to previous node data field

and the address of the link Part is stored is now stored in ‘p’ pointer.

For i= 1 to I < pos by 1

Prev = p

P = p 🡪 link

[ end of for loop ]

Delete the ‘p’ pointer which is pointing to current node after

establishing the link between prev and p pointer

Prev->link=p->link

[ end of else ]

**Counting the no. of nodes**

Step 1: Declare and initialize pointer ‘p ’ to first node

[ p = h ]

Step 2: initialize count = 0

Step 3: if p = = NULL

Write “list is empty “

else

Step 4: While ( p!=NULL) do Step 5

Step5: Count 🡨 count + 1

P = p 🡪link

Return count

[end of else]

**Main function:**

Create an object of the class, accept the choice from the user and call the functions in order to perform the operations according to users choice.

**Program :**

#include <stdio.h>

#include<process.h>

#include<stdlib.h>

#include<conio.h>

struct node

{

int data;

struct node \*link;

}\*h=NULL;

typedef struct node NODE;

void create();

void insert();

void disp();

int count();

void del();

void create()

{

NODE \*t,\*p;

int i,n,num;

p = h;

printf("\n Enter the number of elements to be added:");

scanf("%d",&n);

for(i = 1;i<=n;i++)

{ printf("enter the data");

scanf("%d",&num);

t = (struct node\*)malloc(sizeof(struct node));

t->data=num;

if(p == NULL)

{

h = t;

t->link = h;

p = h;

}

else

{

while(p->link != h)

p = p->link;

t->link = h;

p->link = t;

}

}

printf("\n The linked list is: ");

disp();

}

void insert()

{

int i, pos, num;

NODE \*p,\*q,\*t;

i = count();

printf("\n Enter the location: ");

scanf("%d",&pos);

if(pos < 1 || pos > i+1)

printf("\n Invalid location ");

else

{

printf("\n The linked list before insertion :");

disp();

t = (struct node\*)malloc(sizeof(struct node));

printf("\n Enter the value:");

scanf("%d",&num);

t->data = num;

p = q = h;

if(pos == 1)

{

h=t;

if (p != NULL)

t->link = p;

else

t->link = h;

p = q;

if(p != NULL)

{

while(p->link != q)

p=p->link;

p->link = h;

}

}

else

{

if(pos == i+1)

{

while(p->link!=h)

p=p->link;

p->link=t;

t->link=h;

}

else

{

for(i=1;i<pos-1;i++)

p = p->link;

t->link=p->link;

p->link=t;

}

}

printf("\n Linked list after insertion:");

disp();

}

}

void del()

{

int i,pos;

NODE \*p,\*prev,\*q;

i = count();

if(i==0)

printf("\n Linked list is empty");

else

{

printf("\n Enter the position");

scanf("%d",&pos);

if(pos<1 || pos >i)

printf("\n Invalid position");

else

{

printf("\n The linked list before deletion:");

disp();

p=q=h;

if((pos==1) && count() == 1)

{

h=NULL;

free(p);

}

else

{

if(pos == 1)

{

h = p->link;

while(p->link != q)

p=p->link;

p->link = h;

free(q);

}

else

{

for(i=1;i<pos;i++)

{

prev=p;

p=p->link;

}

prev->link=p->link;

free(p);

}

}

printf("\n Linked list after deletion :");

disp();

}

}

}

void disp()

{

NODE \*p=h;

if(p==NULL)

printf("\n Linked list is empty");

else

{

do

{

printf("%d",p->data);

p=p->link;

}while(p!=h);

}

}

int count()

{

NODE \*p=h;

int cnt=0;

if(p==NULL)

return 0;

else

{

do

{

cnt++;

p=p->link;

}while(p!= h);

return cnt;

}

}

void main()

{

int c=0;

clrscr();

do

{

printf("\n 1.Create");

printf("\n 2.Insert");

printf("\n 3.Delete");

printf("\n 4.Display");

printf("\n 5.Exit");

printf("\n Enter your choice: ");

scanf("%d",&c);

switch(c)

{

case 1:

create();

break;

case 2:

insert();

break;

case 3:

del();

break;

case 4:

if(count() == 0)

printf("\n Linked list is empty");

else

disp();

break;

case 5:

exit(0);

}

}while(c!=5);

getch();

}

**Output:**

1. Create

2. Insert

3. Delete

4. Display

5. Exit

Enter your choice: 4

Linked list is empty

1. Create

2. Insert

3. Delete

4. Display

5. Exit

Enter your choice: 1

Enter the number of elements to be added 4

Enter the 1 data : 10

Enter the 2 data : 15

Enter the 3 data : 20

Enter the 4 data : 30

The linked list is 10 15 20 30

1. Create

2. Insert

3. Delete

4. Display

5. Exit

Enter your choice: 2

Enter the position: 1

The linked list before insertion : 10 15 20 30

Enter the value 5

Linked list after insertion 5 10 20 25 30

1. Create

2. Insert

3. Delete

4. Display

5. Exit

Enter your choice: 3

Enter the position 3

The linked list before deletion: 5 10 20 25 30

Linked list after deletion : 5 10 25 30

1. Create

2. Insert

3. Delete

4. Display

5. Exit

Enter your choice: 4

5 10 25 30

1. Create

2. Insert

3. Delete

4. Display

5. Exit

Enter your choice:5

**Program 6: A menu driven program to perform operations on a stack (linked list**

**implementation).**

**Algorithm :**

Step 1:The data part of the node and ‘link’ pointer of node type is being declared

within the structure a pointer named ‘ top ‘ is being declared in the class stack

is initialized to NULL with the help of constructor and various functions are declared within the class and defined outside the class separately which are used

in main function. The display function is used to display the elements in the stack.

The functions used are as follows:

**void push( )**

**void pop( )**

**void disp( )**

**void stacktop( )**

**Function definitions:**

**1.Push ( ):**

Step 1: Declare a pointer of node type

Step 2: Accept the value

Step 3: Allocate the memory for the node using malloc function

and pointer ‘t’ stores the address of node

newnode= (node \*) malloc (sizeof(node))

Step 4: Insert the element into the data field of the node pointed by newnode

newnode🡪data=item

Step 5: newnode->link=top

Step 6:top=newnode

[ end of function]

**2.pop ( )**

Step 1: declare the pointer currnode of node type

Step 2: if (top = = NULL)

Write : “stack underflow”

else

currnode= top

ele=currnode->info

top=currnode->link

[end of else structure ]

[end of function]

**3.Stacktop ( )**

Step 1:

If (top = = NULL)

Write “Stack underflow “

Else

Display the top element pointed by ‘top’ pointer.

top🡪info

**Main function:**

Accept the choice from the user and call the functions

in order to perform the operations according to users choice.

**Program :**

/\*

\* 6. Write a menu driven program to perform operations on a stack (linked list implementation)

\*/

#include <stdio.h>

#include <stdlib.h>

struct node

{

int info;

struct node \*ptr;

}\*top,\*top1,\*temp;

int count = 0;

/\* Create empty stack \*/

void create()

{

top = NULL;

}

/\* Push data into stack \*/

void push(int data)

{

if (top == NULL)

{

top =(struct node \*)malloc(1\*sizeof(struct node));

top->ptr = NULL;

top->info = data;

}

else

{

temp =(struct node \*)malloc(1\*sizeof(struct node));

temp->ptr = top;

temp->info = data;

top = temp;

}

count++;

}

/\* Pop Operation on stack \*/

void pop()

{

top1 = top;

if (top1 == NULL)

{

printf("\n Error : Trying to pop from empty stack");

return;

}

else

top1 = top1->ptr;

printf("\n Popped value : %d", top->info);

free(top);

top = top1;

count--;

}

/\* Display stack elements \*/

void display()

{

top1 = top;

if (top1 == NULL)

{

printf("Stack is empty");

return;

}

while (top1 != NULL)

{

printf("%d\n", top1->info);

top1 = top1->ptr;

}

}

int main()

{

int no, ch, e;

printf("\n 1 - Push");

printf("\n 2 - Pop");

printf("\n 3 - Dipslay");

printf("\n 0 - Exit");

create();

while (1)

{

printf("\n Enter choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("Enter data : ");

scanf("%d", &no);

push(no);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 0:

exit(0);

default :

printf(" Wrong choice!, Please enter correct value.");

break;

}

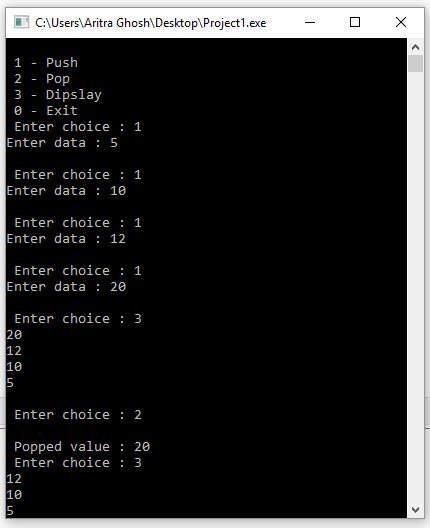
}

return 0;

system("PAUSE");

}

**Output:**



**Program 7: A menu driven recursive program to a) Find factorial of a given number b) Generate first N terms of a fibonacci c) GCD of three numbers**.

**Algorithm :**

Step 1: **declaration of functions**

long fact (int)

int gcd (int,int)

int fibo (int)

**Main function:**

Step 1: Declare the variables ch ,f ,n ,t, i and initialize ch = 0 and f =0

Step 2: accept choice from user

do

Step 3 until choice entered by user is ( ch < = 3 )

Choice 1: Accept any positive number

**call the function fact**

Choice 2: Accept the number of elements in series within for loop

for i=1 to i<=n by 1

**call the function fibo (i)**

read the data element**s**

Choice 3:

Accept any 3 values to variables a, b, c

If(a==0||b==0||c==0)

Write ”invalid input”

Else

Call the function gcd (a,b) and assign to t

t = gcd (a,b)

call the function gcd(t, c) and assign to t

t=gcd (t,c)

[end of main function ]

**Function definitions**

**1.long fact ( int n)**

Step 1:

if ( n = = 0)

return 1

Step 2:

else

return ( n \* (fact(n-1)

[ end of function ]

**2. int fibo (int i)**

Step 1:

if ( i = = 1 )

return 0

Step 2:

if ( i= = 2 )

return 1

Step 3:

else

return fibo (i-1) + fibo (i-2)

[end of else structure]

[end of function]

**3.int gcd (int x,int y)**

Step1: if (y==0)

return x

Step2: if(x< y)

return(gcd (y ,x))

else

return (gcd (y, x % y))

**Program :**

//Write a menu driven recursive program to

//a) find factorial of a given number

//b) generate first N terms of a fibonacci sequence

//c) GCD of three numbers.

#include <stdio.h>

#include <stdlib.h>

long factorial(int n){

if(n>=1)

return n\*factorial(n-1);

return 1;

}

int fibonacciSeris(int n)

{

if ( n == 0 )

return 0;

else if ( n == 1 )

return 1;

else

return (fibonacciSeris(n-1) + fibonacciSeris(n-2));

}

int gcd(int a, int b)

{

if (a == 0)

return b;

return gcd(b%a, a);

}

int getGCD(int arr[])

{

int i,result = arr[0];

for(i = 1; i < 4; i++)

result = gcd(arr[i], result);

return result;

}

int main() {

int number,arr[4];

int ch,i,c;

long result;

while(1){

printf("Factorial of a Number: [PRESS 1]\n");

printf("Show first N terms of a fibonachi Sequence: [PRESS 2]\n");

printf("GCD of three Numbers: [PRESS 3]\n");

printf("Exit: [PRESS 0]\n");

scanf("%d", &ch);

switch(ch){

case 1:

printf("\n\n\*\*\*\*\*Factorial of a Number\*\*\*\*\*\n\n");

printf("Enter number: ");

scanf("%d", &number);

result = factorial(number);

printf("Facrotial of %d: %d! = %ld\n\n",number,number,result);

break;

case 2:

printf("\n\n\*\*\*\*\*Fibonachi Seris\*\*\*\*\*\n\n");

printf("How long you want? ");

scanf("%d", &number);

c = 0;

printf("Fibonacci series\n");

for(i= 1 ; i<=number ; i++)

{

printf("%d\t", fibonacciSeris(c));

c++;

}

printf("\n\n");

break;

case 3:

printf("\n\n\*\*\*\*\*GCD of three numbers\*\*\*\*\*\n\n");

printf("Enter three numbers\n");

for(i = 0; i<3; i++)

scanf("%d", &arr[i]);

number = getGCD(arr);

printf("GCD = %d\n",number);

break;

case 0:

exit(1);

system("PAUSE");

default:

printf("Wrong Choice !\n\n");

system("PAUSE");

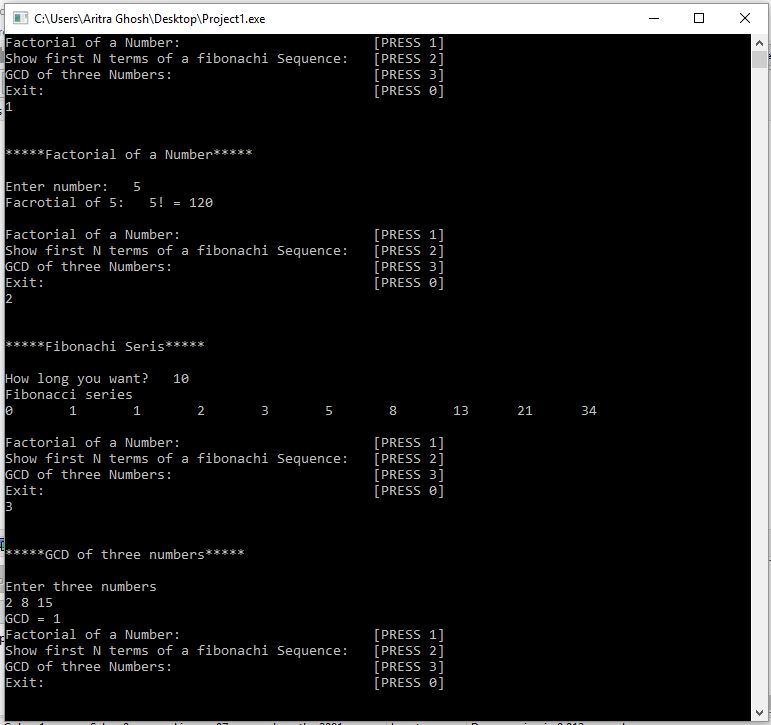
}

}

return 0;

}

**Output :**



**Program 8: A program to solve the problem of towers of Hanoi with 3 pegs and n**

**discs**

**Algorithm :**

Step 1: Declare the variables and function with source, aux, dest as the

variables of character type and a variable ‘n’ is of integer type, the function used is as follows:

void tower( int, char, char ,char)

**Function definition:**

**tower ( int n , char source, char dest, char aux )**

Step 1:

if ( n = = 1)

Write: “Move source disk to destination disk“

Step 2:

Else

**Call the function: tower (n-1,source,aux,dest)**

Write: “Move source disk to destination disk”

**Call the function: tower (n-1,aux,dest,source)**

return

[end of else structure ]

[ end of function ]

**Main function:**

Accept the number of disks and call the function by passing the disks accepted as parameters.

**Program :**

//Q.8: A program to solve the problem of towers of Hanoi with 3 pegs and n discs

#include <stdio.h>

// C recursive function to solve tower of hanoi puzzle

void towerOfHanoi(int n, char from\_rod, char to\_rod, char aux\_rod)

{

if (n == 1)

{

printf("\n Move disk 1 from rod %c to rod %c", from\_rod, to\_rod);

return;

}

towerOfHanoi(n-1, from\_rod, aux\_rod, to\_rod);

printf("\n Move disk %d from rod %c to rod %c", n, from\_rod, to\_rod);

towerOfHanoi(n-1, aux\_rod, to\_rod, from\_rod);

}

int main()

{

int n;

printf("Howmmany Disks? ");

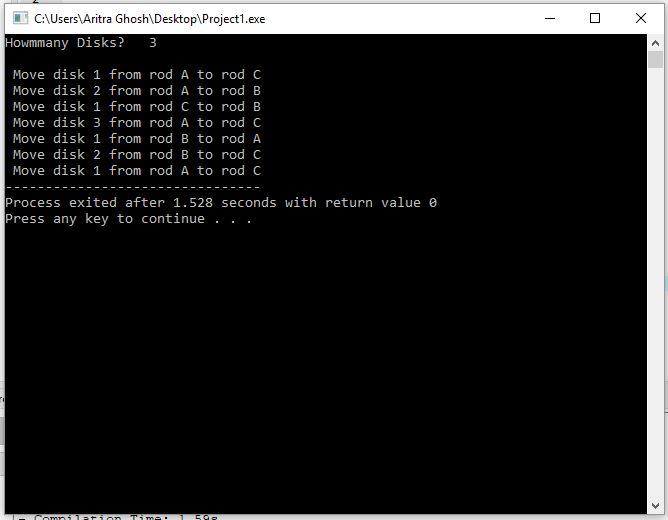
scanf("%d",&n);

towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods

return 0;

}

**Output:**



**Program 9: : A menu driven program to perform operations on a circular**

**queue(linked list implementation).**

**Algorithm :**

Step 1: The data part and the pointer ‘next’ is declared within a structure of node type,

two pointers ‘front’,’ rear’ of the node type and the required functions are declared within the class called cqueue.

**Functions:** void enqueue ( )

void dequeue ( )

void show( )

void q\_front ( )

void q\_rear ( )

**Function declaration:**

**1. enqueue ( )**

Step 1: Declare variable ‘value’ of integer type and a pointer ‘p’ of node type

Step 2: Accept the value

Step 3: Allocate the memory for the node using malloc function and pointer ‘p’

Stores the address of the node

p=(node \*) malloc(sizeof (node))

Step 4: The value accepted is stored in data field whose address is stored in ‘p’

pointer and the info field of the pointer is NULL which is pointed by ‘next’

pointer.

p🡪data = value

p🡪next=NULL

Step 5: if (front= =NULL) then [indicates queue is empty]

front = rear = p

[ both the pointers hold the address of the first node ‘p’]

[end of if]

Step 6:

else rear ->next = p

rear = p [the address of pointer ‘p’ is stored in rear ]

Step 7:

p->next = front

[ the address of ‘front’ is now stored in ‘p’ which is pointed by pointer ‘next’]

[end of function]

2**. q\_front( )**

Step 1: if (front = = NULL)

Write: “Queue is empty”

else

Front ->data [the value of the data field is displayed is queue is

Not empty which is pointed by ‘front’]

[end of if else structure]

[end of function]

3. **q\_rear ( )**

Step 1: if (rear = = NULL)

Write: “queue is empty “

else

rear->data

[ the value of the data field at the rear end is pointed by rear pointer is displayed ]

[end of if else ]

[end of function ]

4. **dequeue( )**

Step 1: if (front = = NULL)

Write: “queue is empty “

Step 2: declare a variable of ‘temp’ of node type and temp variable

holds the address of ‘front’ pointer

temp = front

Step 3: if (front = = rear) then

front = rear = NULL ( both the pointers point to NULL)

temp 🡪 data

free ( temp ) { when both front and rear pointers to NULL, the temp pointer

is stores address of data which is deleted }

else

Step 4:

front = front->next

rear->next = front

temp->data

free ( temp )

[‘front’ pointer holds the address of next node pointed by ‘next’, rear

pointer holds the address of front and the data field address is stored

in temp pointer which is deleted ]

**5. Show ( )**

Step 1: Declare ‘p’ pointer which holds the address of ‘front’ pointer

Step 2:

if ( front = = NULL )

Write: “Queue is empty”

Else

Do until (p ! = front)

P=p->next (display the queue elements)

[ end of do while loop ]

[ end of else ]

**Main function**

Accept the choice from the user and call the functions in order to perform the

operations according to users choice.

**Program :**

//Q.9: Write a menu driven program to perform operations on a circular queue (linked //list implementation).

#include<stdio.h>

#include<stdlib.h>

#define que struct queue

#define pf printf

#define sf scanf

struct queue{

int info;

struct queue \*link;

};

que \*front=NULL,\*rear=NULL;

int count=0;

void push(int n){

que \*newnode;

newnode=(struct queue\*)malloc(sizeof(struct queue));

newnode->info=n;

newnode->link=NULL;

if(count==0)

front=newnode;

else

rear->link=newnode;

rear=newnode;

rear->link=front;

count++;

}

int pop(void){

int n;

que \*temp;

if(count==0)

return (-1);

count--;

if(front==rear)

{

n=front->info;

free(front);

front=NULL;

rear=NULL;

}else

{

temp= front ;

n = temp-> info ;

front = front -> link ;

rear -> link = front ;

free ( temp ) ;

}

return n;

}

void display(void){

que \*temp;

int i;

if(count==0)

pf("Empty");

else

{

temp=front;

for(i=0;i<count;i++)

{

pf("%d ",temp->info);

temp=temp->link;

}

}

pf("\n");

}

int size(void)

{

return count;

}

int main()

{

int n,ch=10;

while(ch!=0)

{

pf("\n What do you want to do??\n");

pf("1.Push\n");

pf("2.Pop\n");

pf("3.SizeOfQueue\n");

pf("4.Display\n");

pf("0.EXIT\n");

sf("%d",&ch);

switch(ch)

{

case 1:

{

pf("What no. do you want to push in queue\n");

sf("%d",&n);

push(n);

break;

}

case 2:

{

n=pop();

if(n==-1)

pf("Queue is empty\n");

else

pf("Number poped from queue is %d\n",n);

break;

}

case 3:

{

n=size();

pf("Size of queue is %d\n",n);

break;

}

case 4:

{

pf("Queue is -->> ");

display();

}

case 0:

break;

default:

pf("Wrong Choice\n");

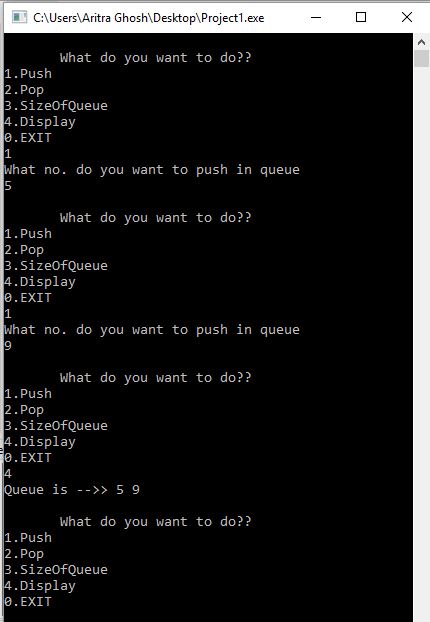
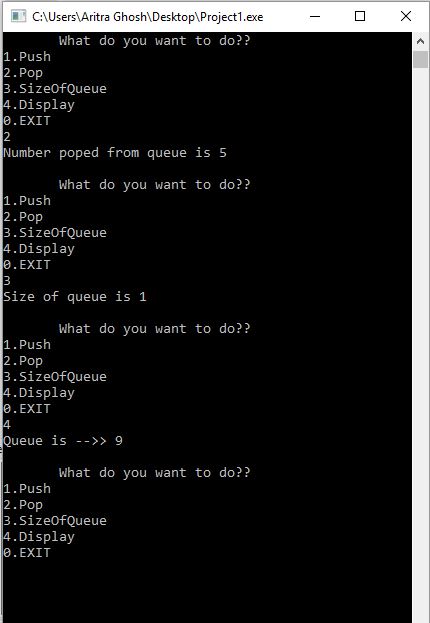
break;

}

}

}

**Output :**



**Program 10: program to find**

**a) length of string**

**b) concatenate two strings**

**c) extract substring from given string**

**d) finding and replacing string by another**

**Algorithm :**

Step 1: declaration of functions, the functions which are used are as

follows

void concat(char\*,char\*)

int length(char \*)

void substring(char \*, int,int)

int stfind(char \*,char \*)

void strep(char\*,char\*,int)

**Main function**

Step 1: declare the variable str1, str2,str3 which stores the character array of size 30. Also declare choice,len,pos,num of type int.

Step 2: Accept the choice from user

Choice 1: Accept two strings str1 and str2 and concatenate both strings using the function concat

**concat(str1,str2)**

Choice 2: Find the length of the of the string, accept the string by accepting a string str1and length of str1 is found out using the string function strlen, and assigned to variable len

**len = strlen(str1)**

Choice 3: Accept the string str1, and enter the position from which extraction should happen (pos). Alos enter thenumber of characters to be extracted(num). find out the length of str1.

If (pos+num-1)>len) then extraction is not possible. Otherwise call the function substring(str,pos,num)

Choice 4: Accept the two strings, find out the position using the function

stfind and assign to variable pos, then the condition is checked for

replacement of string.

**pos= stfind(str1,str2)**

if (pos >0) then

accept the third string to be replaced i.e str2

replace the string by calling the function strep along with

the position

**strep(str1,str2,pos)**

[end of if structure]

else

Substring is not found

**Function definitions:**

**Substring ()**

Step 1. Declare variable I of type integer ant as a character pointer of type char.

Step 2: t🡨ptr+p-1

Step 3: for i=0 to n by 1

\*(t+i🡨\*(ptr+p-1+i)

\*(t+i)🡨’\0’

[end of for]

2. **length( )**

Step 1: initialize len=0;

[ string is read until NULL value is reached ,if string is not equal to NULL

Perform the operations specified within while loop i.e increment the pointer p

to read next character and increment the variable len and finally return value

of the len]

Step 2: while(\*ptr!='\0') do step 3

Step3:

ptr🡨ptr+1

len🡨len+1

[end of while loop]

return len

3. **concat()**

Step 1: Two parameters p1,p2 of character pointer are passed to the function

Step 2: Check whether ‘p1’ not equal to NULL, increment p1 to next

character position.

While(\*p1!='\0')

p1🡨p1 + 1

Step 3: check for whether ‘p2’ not equal to NULL, if not then continue with

step 4

while(\*p2!='\0') do

Step 4:

\*p1🡨\*p2 [contents of pointer ‘r2’ stored in pointer ‘r1’]

p1🡨p1 + 1

p2🡨p2+1

\*r1='\0' [ pointer r1 initialized to NULL ]

[End of while ]

Step 5: write t as the extracted string.

4. **stfind:**

Step 1: declare the variables len,len2,i,j,k

[initialize i =0 ]

len 🡨strlen (r1) [find the length of the string pointed by r1 and assign to

len]

len2🡨strlen(r2) [find the length of the string pointed by r2 and assign to

len2]

Step 2: for i=0 to len-len2+1 by 1

K=1

Pos=pos+1

for j = 0 to j<len2 to by

if ( \*(p1+i+j) != \*(p2+j) [check for contents of p1 and p2]

k=0

[end of for loop]

Else k=2

If(k=2)

Return pos

[end of for loop]

5. **strep ( ):**

Step 1: declare two variables i,len

Step 2: find out the length of the string pointed by pointer p3

assign to variable len

len 🡨strlen(p3)

Step 3:

for i=pos-1 to i<len+pos-1 by 1

\*(p1+1) 🡨\*p3 [contents of r3 is assigned to r1 content]

p3 🡨p3 + 1 [increment the pointer to next character position]

[end of for loop ]

**Program :**

//10. Write a menu driven program to ......

//a) find the length of a string

//b) concatenate two strings

//c) to extract a substring from a given string

//d) finding and replacing a string by another string in a text ( Use pointers and user-defined functions)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int string\_length(char arr[]){

int i = 0;

while(arr[i] != '\0'){

i++;

}

return i;

}

void string\_concatinate(char str1[], char str2[]){

int i = 0,j = 0;

while(str1[i] != '\0'){

i++;

}

while(str2[j] != '\0'){

str1[i] = str2[j];

i++;

j++;

}

str1[i] = '\0';

}

char\* extract\_substring(char\* str, int start, int end, int len)

{

char\* result;

int i,count=0;

if (str == 0 || strlen(str) == 0 || strlen(str) < start || strlen(str) < (start+end))

return 0;

else

{

result = (char \*)malloc(sizeof(len));

for(i = start-1, count = 0; i<=end; i++, count++){

result[count]= str[i];

}

result[i] = '\0';

return result;

}

}

char \*string\_replace(char \*str, char \*str1, char \*str2){

char \*result;

int len1, len2, i, count = 0;

len1 = strlen(str1);

len2 = strlen(str2);

for(i = 0; str[i] != '\0'; i++){

if(strstr(&str[i], str1) == &str[i])

{

count++;

i += len1-1;

}

}

result = (char \*)malloc(i + count \* (len2 - len1) + 1);

i = 0;

while(\*str)

{

if(strstr(str, str1) == str)

{

strcpy(&result[i], str2);

i += len2;

str += len1;

}

else

result[i++] = \*str++;

}

result[i] = '\0';

return result;

}

int main() {

int ch;

int start, end;

char str[100],str2[100],str1[100];

char \*result = NULL;

int length,len;

while(1){

printf("Find the length of a String: [PRESS 1]\n");

printf("Concatenate two strings: [PRESS 2]\n");

printf("Extract a substring from a given string: [PRESS 3]\n");

printf("finding and replacing a string: [PRESS 4]\n");

printf("Exit: [PRESS 0]\n");

scanf("%d", &ch);

switch(ch){

case 1:

printf("\n\n\*\*\*\*\*Length of a String \*\*\*\*\*\n");

printf("Enter a string: \n");

scanf("%s",str);

length = string\_length(str);

printf("Length of String : '%s' = %d\n\n", str,length);

break;

case 2:

printf("\n\n\*\*\*\*\*Concatinate two Strings \*\*\*\*\*\n");

printf("Enter first string: \n");

scanf("%s",str);

strcpy(str1,str);

printf("Enter Secand string: \n");

scanf("%s",str2);

string\_concatinate(str, str2);

printf("Concatination of : '%s' & '%s' :\t%s \n",str1, str2, str);

break;

case 3:

printf("\n\n\*\*\*\*\*Extract a sustring from a Strings \*\*\*\*\*\n");

printf("Enter a string: \n");

scanf("%s",str);

len = string\_length(str);

printf("\nEnter Start: ");

scanf("%d", &start);

printf("\nEnter End: ");

scanf("%d", &end);

result = extract\_substring(str, start, end, len);

if(result != 0)

printf("Substring of String : '%s' = '%s'\n\n", str,result);

else

printf("\n\nString Extraction Not possible.\n");

break;

case 4:

printf("\n\n\*\*\*\*\*String Replace with Strings \*\*\*\*\*\n");

printf("Enter a String: \n");

scanf("%s",str);

printf("Enter string to find: \n");

scanf("%s",str1);

printf("Enter string to replace: \n");

scanf("%s",str2);

result = string\_replace(str, str1, str2);

printf("Replaced String: '%s'\n\n", result);

break;

case 0:

printf("Thanks ! EXIT \n\n");

exit(1);

system("PAUSE");

default:

printf("Wrong Choice !\n\n");

system("PAUSE");

}

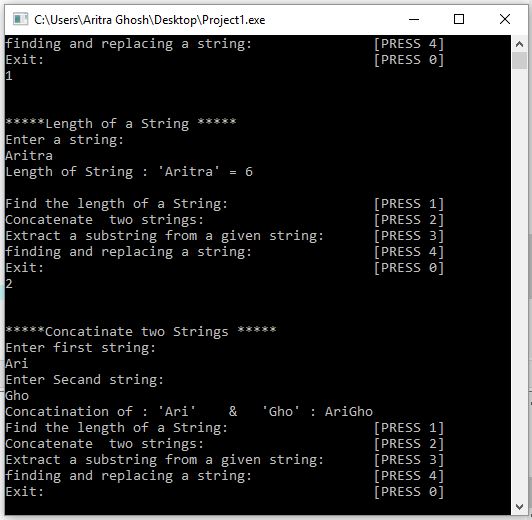
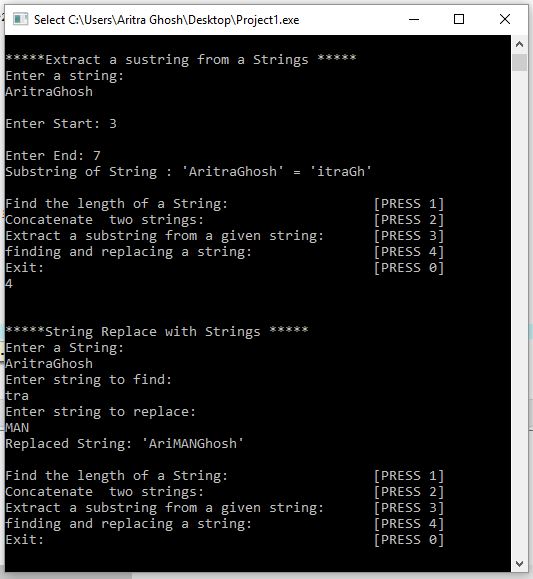
}

system("PAUSE");

return 0;

}

**Output :**



**Program 11: A program to convert the given infix expression into its postfix form.**

**Algorithm :**

Step 1: Character array variables are being declared which stores

the constant size, the variables declared are

( infix[size], postfix[50], stack[50] ,top=0)

Step 2: stack[0] 🡨'(';

Step 3: read infix

Step 4: j🡨strlen(infix)

Step 5: for i=0 to n by 1

ch🡨infix[i]

if(((ch>='0')&&(ch<='9'))||((ch>='a')&&(ch<='z'))||((ch>='A')&&(ch<='Z')))

write ch

[end if]

if(ch=='(')

top🡨top+1

stack[top] 🡨ch

[end if]

if(ch=='^')

while(stack[top]=='^') do

top🡨top-1

write stack[top]

[end of while]

Top🡨top+1

stack[top] 🡨ch

[end if]

if((ch=='\*')||(ch=='/'))

while((stack[top]=='\*')||(stack[top]=='/')||(stack[top]=='^')) do

top🡨top-1

write stack[top]

[end of while]

Top🡨top+1

stack[top] 🡨ch

[end if]

if((ch=='+')||(ch=='-'))

while((stack[top]=='\*')||(stack[top]=='/')||(stack[top]=='+')||(stack[top]=='-')||(stack[top]=='^'))

write stack[top]

top🡨top+1

[end of while]

Top🡨top+1

stack[top] 🡨ch

[end if]

if(ch==')')

while(stack[top]!='(')

top🡨top-1

write stack[top]

[end of while]

top🡨top-1

[end if]

[end for]

Step 6: while(stack[top]!='(') do

top🡨top-1

write stack[top]

[end of while]

**Program :**

//11. Write a program to convert the given infix expression into its postfix form.

#include<stdio.h>

#include<ctype.h>

char stack[20];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

}

main()

{

char exp[20];

char \*e, x;

printf("Enter the expression :: ");

scanf("%s",exp);

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c",pop());

push(\*e);

}

e++;

}

while(top != -1)

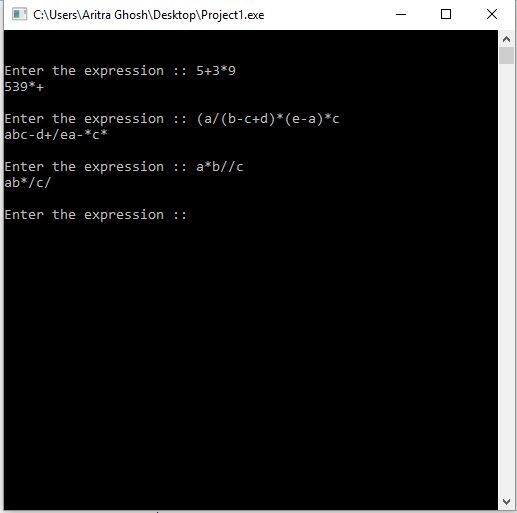
{

printf("%c",pop());

}

}

**Output :**



**Program 12: Program to evaluate postfix expression**

**Algorithm :**

Step 1: Declare variables len, stack[50],len,value,n1,n2,finalresult,result

**function definition:**

1. **push ( int item)**

Step 1: top 🡨 top + 1

s[top] 🡨 item

[ increment the top variable and assign the symbol to top element of stack]

2. **float pop ( )**

Step 1 : if(top==0)

Write invalid post fix expression

top🡨top-1

stack[top]=0

return

**Main function:**

Step 1 : Accept the postfix expression

Step 2 : find the length of the of the postfix expression and assign to len

Step 3 :for i=0 to i<len by 1

if ( postfix > = 0) && postfix< = ‘9’) then

**call function**

push (postfix– ‘0’)

[end of if ]

else

if((postfix>='a'&&postfix<='z')||(postfix>='A'&&postfix<='Z'))

read the value for postfix

push(value)

[end of if ]

else

n1🡨pop()

n2🡨pop()

switch(postfix)

case '^' :

result🡨pow(n1,n2)

push(result)

case '\*' :

result🡨n2\*n1

push(result)

case '/' :

result🡨n2/n1

push(result)

case '+' :

result🡨n2+n1

push(result)

case '-' :

result🡨n2-n1

push(result)

default: write invalid postfix expression

[end of switch]

[end of else]

[end of for]

Step 4:finalresult🡨pop()

Step 5: Write final result as thevalue of the expression

**Program :**

#include<stdio.h>

#define MAX 20

typedef struct stack

{

int data[MAX];

int top;

}stack;

void init(stack \*);

int empty(stack \*);

int full(stack \*);

int pop(stack \*);

void push(stack \*,int);

int evaluate(char x,int op1,int op2);

int main()

{

stack s;

char x;

int op1,op2,val;

init(&s);

printf("Enter the expression(eg: 59+3\*)\nSingle digit operand and operators only:");

while((x=getchar())!='\n')

{

if(isdigit(x))

push(&s,x-48); //x-48 for removing the effect of ASCII

else

{

op2=pop(&s);

op1=pop(&s);

val=evaluate(x,op1,op2);

push(&s,val);

}

}

val=pop(&s);

printf("\nValue of expression=%d",val);

return 0;

}

int evaluate(char x,int op1,int op2)

{

if(x=='+')

return(op1+op2);

if(x=='-')

return(op1-op2);

if(x=='\*')

return(op1\*op2);

if(x=='/')

return(op1/op2);

if(x=='%')

return(op1%op2);

}

void init(stack \*s)

{

s->top=-1;

}

int empty(stack \*s)

{

if(s->top==-1)

return(1);

return(0);

}

int full(stack \*s)

{

if(s->top==MAX-1)

return(1);

return(0);

}

void push(stack \*s,int x)

{

s->top=s->top+1;

s->data[s->top]=x;

}

int pop(stack \*s)

{

int x;

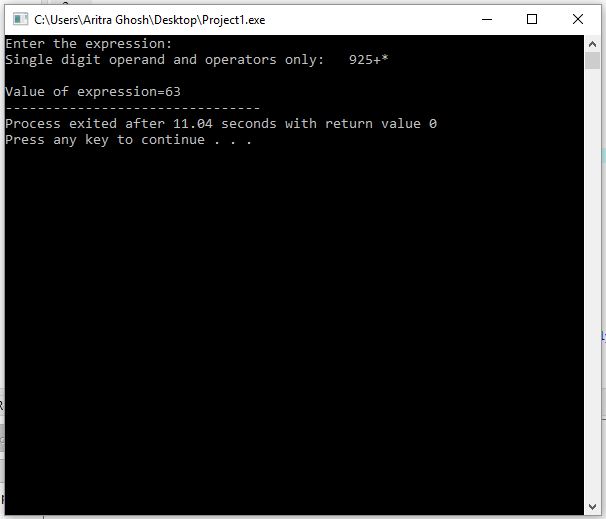
x=s->data[s->top];

s->top=s->top-1;

return(x);

}

**Output:**



**Program 13: A menu driven program to create a binary tree and to perform insert**

**and delete operations.**

**Algorithm :**

Step 1: Start

Step 2: Create a structure and declare node \*link and \*rlink with it along with data.

Step 3: Declare \*n, \*t, s, d

Step 4: Initialize head = NULL

Step 5: Accept the choice of operation from user store in s

Step 6: Execute corresponding functions from according to choice

Step 7: finsert ()

Declare \*n

Accept value of h -> data

n->left=NULL;

n->right=NULL;

return n;

**Creation of linked list**

Step 1: Initially ‘h’ pointer points to NULL, indicating node is empty

Step 2: Another pointer ‘p’ points to first node i.e ‘h’

[Initialize p:=h]

Step 3: create a new node pointed by ‘p’ pointer

Step 4: Read in the data element and store the data field

t->data=num

t->link=NULL

Step 5: If (p = = NULL), then this new node is first node

h🡨t

p🡨h

else

p = p --🡪 link

p🡪link = t

Step 6: insert (node \* h)

Declare \*t,\*n;

Initialize t=h;

Accept value of n->data

n->left=NULL;

n->right=NULL;

while(t->left!=NULL || t->right!=NULL)

if(t->left!=NULL)

if(n->data < t->data)

t=t->left;

Step 7: Check if (t->right!=NULL)

Check if if(n->data>=t->data)

t=t->right;

if((t->left==NULL) && (n->data < t->data))

break;

if((t->right==NULL) && (n->data >= t->data))

break;

if((n->data < t->data) && (t->left==NULL))

t->left=n;

if((n->data > t->data) && (t->right==NULL))

t->right=n;

Step 8: delete ( )

Declare f=0,f1=0, \*p,\*t,\*t1,\*x;

t=head;

while(t!=NULL)

if(t->data==d)

then

f=1;

x=t;

endif

if(t->data > d)

then

p=t;

t=t->left;

else if(t->data <= d)

then

p=t;

t=t->right;

Step: 9 if(f==0)

Display “Given element not found”

return head;

if(x->left==NULL && x->right==NULL)

if(p->right==x)

p->right=NULL;

else

p->left=NULL;

free(x);

return head;

Step 10: if(x->left !=NULL && x->right!=NULL)

then

p=x;

t1=x->right;

while(t1->left!=NULL)

p=t1; f1=1;

t1=t1->left;

end if

Step: 11 if(t1->left==NULL && t1->right==NULL)

then

x->data=t1->data;

if(f1==1)

p->left=t1->left;

if(f1==0)

x->right=t1->right;

free(t1);

return head;

if(t1->right!=NULL)

x->data=t1->data;

if(f1==1)

p->left=t1->right;

if(f1==0)

p->right=t1->right;

free(t1);

return head;

end if

Step: 12 if(x->left==NULL && x->right!=NULL && x->data!=head->data)

then

if(p->left==x)

p->left=x->right;

else

p->right=x->right;

free(x);

return head;

Step: 13 if(x->left!=NULL && x->right==NULL && x->data!=head->data)

then if(p->left==x)

p->left=x->left;

else

p->right=x->left;

free(x);

return head;

Step: 14 if(x->left!=NULL && x->right==NULL && x->data==head->data)

then head=x->left;

free(p);

return head;

Step:15 if(x->left==NULL && x->right!=NULL && x->data==head->data)

then

head=x->right;

free(p);

return head;

Step: 16 Stop

**Main function:**

Accept the choice from the user and call the

functions in order to perform the operations according to users choice.

**Program :**

//binary tree

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

struct node

{

int info;

struct node \*left;

struct node \*right;

};

typedef struct node NODE;

NODE \*root=NULL;

void create\_tree(NODE \*ptr)

{

NODE \*newleft, \*newright;

int item;

char ch;

if(ptr != NULL)

{

printf("\n enter an element");

scanf("%d",&item);

ptr->info=item;

printf("\n do you want to create a left child of %d :[y/n]\n",ptr->info);

ch=getche();

if(ch=='y' || ch=='Y')

{

newleft=(NODE\*)malloc(sizeof(NODE));

ptr->left=newleft;

create\_tree(newleft);

}

else

{

ptr->left=NULL;

create\_tree(NULL);

}

printf("\n do you want to create right child of %d:[y/n]\n",ptr->info);

ch=getche();

if(ch=='Y'|| ch=='y')

{

newright=(NODE\*)malloc(sizeof(NODE));

ptr->right=newright;

create\_tree(newright);

}

else

{

ptr->right=NULL;

create\_tree(NULL);

}

}

}

void disp(struct node \*ptr,int level)

{

int i;

if(ptr!=NULL)

{

disp(ptr->right,level+1);

for(i=0;i<level;i++)

printf(" ");

printf("%2d\n",ptr->info);

disp(ptr->left,level+1);

}

}

void deleteTree(struct node\* node)

{

if(node==NULL)

return;

deleteTree(node->left);

deleteTree(node->right);

printf("\n deleting node: %d",node->info);

}

void main()

{

int item,ch;

clrscr();

root=NULL;

while(1)

{

printf("\nBINARY tree menu");

printf("...................");

printf("\n 1.create \n2.display \n3.deletes \n4.exit ");

printf("\n enter your choice");

scanf("%d",&ch);

switch(ch)

{

case 1:

root=(NODE\*)malloc(sizeof(NODE));

create\_tree(root);

break;

case 2:

printf("\nthe binary tree nodes are :\n\n\n");

disp(root,1);

break;

case 3:

deleteTree(root);

printf("\n complete tree is deleted");

printf("\n create a new tree");

break;

case 4:

exit(1);

break;

default:

printf("invalid choice");

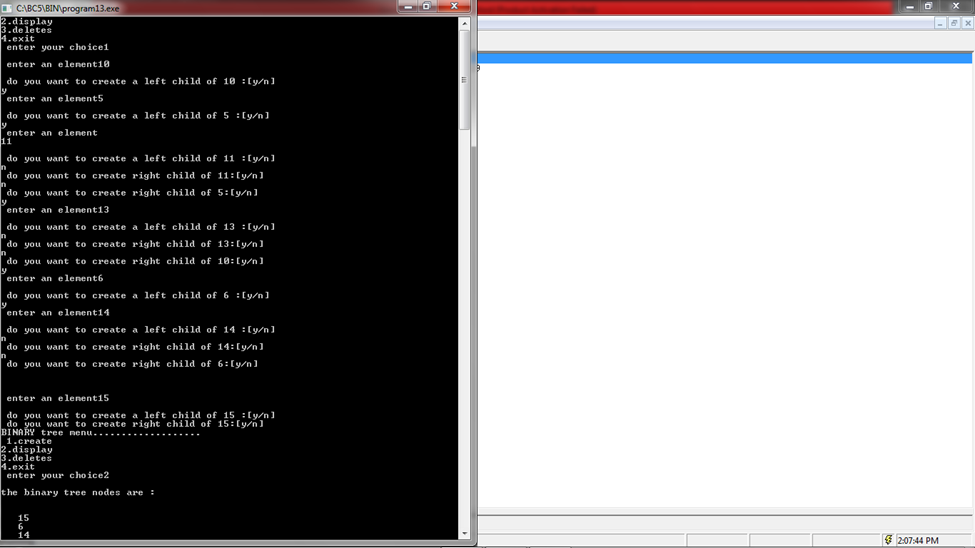
break;

}

}

}

**Output :**





**Program 14: A menu driven program to create a binary search tree and to perform**

**inorder, preorder and post order traversal.**

**Algorithm :**

Step 1: Two pointers llink, rlink of node type and an integer variable data is

being declared

Step 2: Pointer variable ‘root’ is being declared of node type and is initialized to

NULL with help of constructor ,the functions are declared and used within

main function, the functions that are used are as

Follows : void create(int )

void disp ( int )

void inorder (node \* )

void porder ( node \* )

void preorder (node \* )

**function definition:**

1. **create ( )**

Step 1 : declare the pointers temp,currptr, ptr of type node

Step2: temp = ( node \* ) malloc (sizeof (node)

Step 4: temp->info=item

Step 5: Both the left and right children of the tree is indicated as NULL

which indicates that both the left and right children are empty

t->llink =NULL

t->rlink=NULL

Step 6: if ( root = = NULL) (indicates that root is empty)

root = temp ( the address of first node is stored in ‘root’ pointer )

[end of if]

else

currptr = root ( the address of root node is stored in ‘currptr’ )

while(currptr!=NULL)

ptr=currptr

currptr=(item>currptr->info)?currptr->rlink:currptr->llink;

[end of while]

if(ptr->info<item)

ptr->rlink=temp

else

ptr->llink=temp

[end of else]

**2. preorder (node \*ptr)**

Step 1: if (ptr ! = NULL)

Write ptr->info

**Call functions** :

preorder(ptr->llink)

preorder(ptr->rlink)

[end of if ]

[end of function ]

4. **inorder (node \*ptr)**

Step 1: if(ptr!=NULL) ( ‘tree is empty”)

**call functions :** inorder (ptr->llink)

ptr ->data ( “display the elements pointed by ptr pointer )

[ end of if ]

[ End of function ]

5. **postorder (node \*ptr)**

Step 1: if ( ptr!=NULL)

**Call functions:**

Postorder (ptr->left)

Postorder (ptr->right)

ptr->data (“ display the elements pointed by ptr pointer )

[end of if ]

[end of function ]

**Main function:**

create an object of the class, accept the choice from the user

and call the functions inorder to perform the operations according to users choice.

**Program :**

//binary search tree

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

struct node

{

struct node \*left;

struct node \*right;

int info;

};

typedef struct node NODE;

NODE \*root=NULL;

void create(int item)

{

NODE \*newnode,\*currptr,\*ptr;

newnode=(NODE\*)malloc(sizeof(NODE));

newnode->info=item;

newnode->left=NULL;

newnode->right=NULL;

if(root==NULL)

root=newnode;

else

{

currptr=root;

while(currptr!=NULL)

{

ptr=currptr;

currptr=(item>currptr->info)?currptr->right:currptr->left;

}

if(item<ptr->info)

ptr->left=newnode;

else

ptr->right=newnode;

}

}

NODE \*search(NODE \*temp,int item)

{

if(temp==NULL)

return NULL;

else if(item<temp->info)

search(temp->left,item);

else if(item>temp->info)

search(temp->right,item);

else

return temp;

}

void pre\_order(NODE \*ptr)

{

if(ptr)

{

printf("%d",ptr->info);

pre\_order(ptr->left);

pre\_order(ptr->right);

}

}

void in\_order(NODE \*ptr)

{

if(ptr)

{

in\_order(ptr->left);

printf("%d",ptr->info);

in\_order(ptr->right);

}

}

void post\_order(NODE \*ptr)

{

if(ptr)

{

post\_order(ptr->left);

post\_order(ptr->right);

printf("%d",ptr->info);

}

}

int main()

{

int item,ch,i,n;

while(1)

{

printf("\n\t binary search tree");

printf("\n.....................\n");

printf("\n1.create BST");

printf("\n2.display in preorder");

printf("\n3.display in inorder");

printf("\n4.display in postorder");

printf("\n5.exit");

printf("\nenter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nenter how many nodes\n");

scanf("%d",&n);

for(i=0;i<n;i++)

{

printf("\n enetr the data for the node\n");

scanf("%d",&item);

create(item);

}

break;

case 2:

printf("\n preorder traversal\n");

pre\_order(root);

break;

case 3:

printf("\n inorder traversal\n");

in\_order(root);

break;

case 4:

printf("\n postorder traversal\n");

post\_order(root);

break;

case 5:

exit(0);

default:

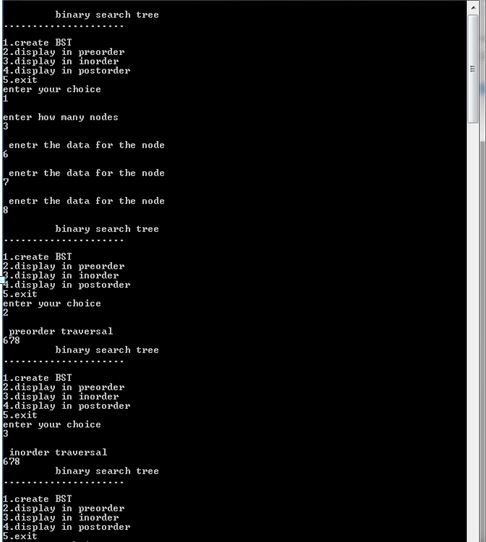
printf("\n invalid choice");

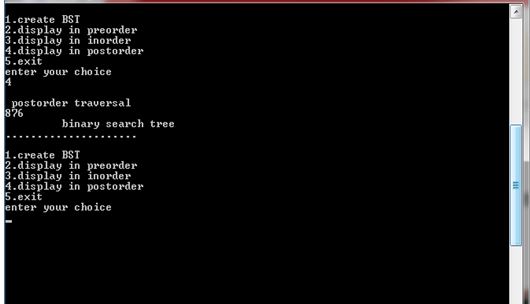
}

}

}

**Output :**





**Program 15: Program to sort N element in ascending order using heap sort**

**Algorithm :**

Step 1: Declare the functions

Functions :

createheap(int[],int);

heapsort(int[],int); (int [ ],int)

function definition:

**1. heapsort(int k[],int n)**

Step 1: declare temp,q,i,j,key of type int;

Step 2:Call createheap(k,n)

Step 3:For q=n;q>=2 decrement q by 1 do

temp🡨k[q]

k[q] 🡨k[1]

k[1] 🡨temp

i🡨1

j🡨2

key🡨k[1]

if((j+1)<q)

if(k[j+1]>k[j])

j🡨j+1

[end of if]

while((j<=(q-1))&&(k[j]>key))

temp🡨k[j]

k[j] 🡨k[i]

k[i🡨temp

i🡨j

j🡨2\*i

if(j+1<q)

if(k[j+1]>k[j])

j🡨j+1

else

if(j>n)

j🡨n

[end of if]

k[i]🡨key

[end of while]

[end of for]

return

2: createheap(int k[],int n)

Step 1: declare temp,q,i,j,key as variables of type int

Step 2:forq=2,q<=n by 1 do

i🡨q

key🡨k[q]

j🡨i/2

while((i>1)&&((key>k[j])))

temp🡨k[j]

k[j]🡨k[i]

k[i]🡨temp

i🡨j

j=🡨/2

if(j<1)

j🡨1

[ end of while]

k[i] 🡨key

[ end of for]

return

**MAIN FUNCTION:**

Step 1 : declare variables n, i ,and array variable k[10]

Step 2: Accept the number of elements

Step 3: For i = 0 to i< x by 1

Accept the elements

[End of for loop ]

Step 4: display the elements by accessing the function heapsort [end of main function ]

**Program :**

**#include<stdio.h>**

**void create(int []);**

**void down\_adjust(int [],int);**

**void main()**

**{**

**int heap[30],n,i,last,temp;**

**printf("Enter no. of elements:");**

**scanf("%d",&n);**

**printf("\nEnter elements:");**

**for(i=1;i<=n;i++)**

**scanf("%d",&heap[i]);**

**//create a heap**

**heap[0]=n;**

**create(heap);**

**//sorting**

**while(heap[0] > 1)**

**{**

**//swap heap[1] and heap[last]**

**last=heap[0];**

**temp=heap[1];**

**heap[1]=heap[last];**

**heap[last]=temp;**

**heap[0]--;**

**down\_adjust(heap,1);**

**}**

**//print sorted data**

**printf("\nArray after sorting:\n");**

**for(i=1;i<=n;i++)**

**printf("%d ",heap[i]);**

**}**

**void create(int heap[])**

**{**

**int i,n;**

**n=heap[0]; //no. of elements**

**for(i=n/2;i>=1;i--)**

**down\_adjust(heap,i);**

**}**

**void down\_adjust(int heap[],int i)**

**{**

**int j,temp,n,flag=1;**

**n=heap[0];**

**while(2\*i<=n && flag==1)**

**{**

**j=2\*i; //j points to left child**

**if(j+1<=n && heap[j+1] > heap[j])**

**j=j+1;**

**if(heap[i] > heap[j])**

**flag=0;**

**else**

**{**

**temp=heap[i];**

**heap[i]=heap[j];**

**heap[j]=temp;**

**i=j;**

**}**

**}**

**}**

**Output :**

