**C Programs**

**\*\*\*\*sayHello.h Header File\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//Don’t Have to compile header file separately. It is compiled with main programs.**

#include <conio.h>

#include <stdio.h>

void sayHello(char name[60] )

{

printf("Welcome %s :",name);

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* dispMsg.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <conio.h>

#include <stdio.h>

#include "sayhello.h"

int main()

{

sayHello("Vaishali");

getch();

return;

}

**\*\*\*\*\*\*Enter Emp Details\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include<conio.h>

#include<stdio.h>

#define TRUE 1;

#define PI 3.14;

//preprocessor before compilation it execute.

int show();

int main()

{

int empid;

char empname[10];

char gender;

float da;

double salary;

printf(" Enter ur EmpId:");

scanf("%d",&empid);

printf(" Enter EmpName:");

scanf("%s",&empname);

fflush(stdin);//for character data

printf(" Enter gender:");

scanf("%c",&gender);

printf(" Enter DA:");

scanf("%f",&da);

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ExepTag.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <conio.h>

#include <stdio.h>

int main()

{ int exep;

printf("Enter ur experience in years ");

scanf("%d",&exep);//speces are not allowed.

if((exep>=0)&&(exep<=3))

{

printf("BLUE");

}

else if((exep>3)&&(exep<=5))

{

printf("GREY");

}

else if((exep>5)&&(exep<10))

{

printf("YELLO");

}

else

{

printf("RED");

}

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Fibo Series\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <conio.h>

#include <stdio.h>

int main()

{

int count, n, t1=0, t2=1, display=0;

printf("Enter number of terms: ");

scanf("%d",&n);

printf("Fibonacci Series: %d+%d+", t1, t2); /\* Displaying first two terms \*/

count=2; /\* count=2 because first two terms are already displayed. \*/

while (count<n)

{

display=t1+t2;

t1=t2;

t2=display;

++count;

printf("%d+",display);

}

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Smallest No in Array\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <stdio.h>

#include <conio.h>

int main()

{

int myarr[100], minimum, size, c, location = 1;

printf("Enter the number of elements in array\n");

scanf("%d",&size);

printf("Enter %d integers\n",size);

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

scanf("%d",&myarr[c]);

minimum = myarr[0];

for (c=1;c<size;c++)

{

if(myarr[c]<minimum)

{

minimum = myarr[c];

location = c+1;

}

}

printf("Minimum element is present at location %d and it's value is %d.\n",location,minimum);

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*enum.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <stdio.h>

#include <conio.h>

enum e\_dept {Accounts,HR,Training};

struct emp

{

int empno;

char ename[10];

enum e\_dept dept;

}e1;

int main()

{

printf("Enter empno:");

scanf("%d",&e1.empno);

printf("\n Enter Name:");

scanf("%s",e1.ename);

e1.dept=Training;

printf("%d %s %d",e1.empno,e1.ename,e1.dept);

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*File Demo.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <stdio.h>

#include <conio.h>

// r+ opens existing file for reading and writing.

//w+ open existing file for reading and writing but overright.

//a++

int main()

{

char ch;

FILE \*fpr,\*fpw;//this is in stdio,h

if((fpr=fopen("ab.txt","r"))==NULL)

printf("File cannot be opened for read");

fpw=fopen("xyz.txt","w");

do

{

ch=getc(fpr);

printf("%c",ch);

putc(ch,fpw);

}while(ch!=EOF);

fclose(fpr);

fclose(fpw);

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*fscanf demo.c\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <stdio.h>

#include <conio.h>

// r+ opens existing file for reading and writing.

//w+ open existing file for reading and writing but overright.

//a++

int main()

{

char ch;

char myBuffer[100];

FILE \*fp;//this is in stdio,h

if((fp=fopen("abc.txt","w"))==NULL)

printf("Not present");

fscanf(stdin,"%[^\n]",myBuffer);//reda from keyboard

fprintf(fp,"%s",myBuffer);//write in file

printf("Reading from file");

if((fp=fopen("abc.txt","r"))==NULL)

printf("No t reading ");

fscanf(fp,"%[^\n]",myBuffer);//reda from file

fprintf(stdout,"%s",myBuffer);//write on console

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*empstructure\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <conio.h>

#include <stdio.h>

struct emp

{

int eid;

char enm[10];

float esal;

struct dob

{

int day;

int mon;

int year;

}empDOB;

};

int main()

{

struct emp e1;

struct emp e2={1234,"abcd",2345.9,{12,12,1290}};

int no;

printf("\n Enter no :");

scanf("%d",&e1.eid);

printf("\n Enter name :");

scanf("%s",e1.enm);//do not use &

printf("\n Enter salary :");

scanf("%f",&e1.esal);

printf("\n Enter DOB");

scanf("%d",&e1.empDOB.day);

scanf("%d",&e1.empDOB.mon);

scanf("%d",&e1.empDOB.year);

printf("EMPID : %d EMPNAME : %s EMPSAL: %f DOB : %d- %d-%d",e1.eid,e1.enm,e1.esal,e1.empDOB.day,e1.empDOB.mon,e1.empDOB.year);

//printf("EMPID : %d EMPNAME : %s EMPSAL: %f DOB : %d- %d-%d",e1.eid,e1.enm,e1.esal,e1.empDOB.day,e1.empDOB.mon,e1.empDOB.year);

//printf("EMPID : %d EMPNAME : %s EMPSAL: %f ",e1.eid,e1.enm,e1.esal);

getch();

return 0;

}

**\*\*\*Write emp details from structure in file\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include<conio.h>

#include<stdio.h>

int main()

{

int empid;

char empname[10];

char gender;

float da;

double salary;

int eid;

char enm[10];

char gen;

float daa;

double sal;

int k;

int count;

int i;

FILE \*fpw;

FILE \* fpr;

fpw=fopen("EmpDetails.txt","a");

fpr=fopen("EmpDetails.txt","r");

printf(" How Many Emp :");

scanf("%d",&count);

/\*

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

{

printf(" \nEnter ur EmpId:");

scanf("%d",&empid);

fflush(stdin);

printf(" Enter EmpName:");

scanf("%s",&empname);

fflush(stdin);//for character data

printf(" Enter gender:");

scanf("%c",&gender);

printf(" Enter DA:");

scanf("%f",&da);

printf(" Enter Salary:");

scanf("%lf",&salary);

printf("\nU Entered : %d %s %c %f %lf",empid,empname,gender,da,salary);

fprintf(fpw,"%d %s %c %f %lf \n",empid,empname,gender,da,salary);

}

fclose(fpw);

printf("\nData is written in File");

\*/

printf("\n\*\*\*\*\*\*\*\*File Read Operation\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

while((fscanf(fpr,"%d %s %c %f %lf",&eid,enm,&gen,&daa,&sal))!=EOF)

{

printf("\n%d%s%c%f%lf",eid,enm,gen,daa,sal);

}

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*Bubble Sort\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**First Pass:**  
( **5** **1** 4 2 8 ) \to( **1** **5** 4 2 8 ), Here, algorithm compares the first two elements, and swaps since 5 > 1.  
( 1 **5** **4** 2 8 ) \to( 1 **4** **5** 2 8 ), Swap since 5 > 4  
( 1 4 **5** **2** 8 ) \to( 1 4 **2** **5** 8 ), Swap since 5 > 2  
( 1 4 2 **5** **8** ) \to( 1 4 2 **5** **8** ), Now, since these elements are already in order (8 > 5), algorithm does not swap them.  
**Second Pass:**  
( **1** **4** 2 5 8 ) \to( **1** **4** 2 5 8 )  
( 1 **4** **2** 5 8 ) \to( 1 **2** **4** 5 8 ), Swap since 4 > 2  
( 1 2 **4** **5** 8 ) \to( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) \to( 1 2 4 **5** **8** )  
Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.  
**Third Pass:**  
( **1** **2** 4 5 8 ) \to( **1** **2** 4 5 8 )  
( 1 **2** **4** 5 8 ) \to( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) \to( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) \to( 1 2 4 **5** **8** )

**\*\*\*\*\*\*\*\*\*\*\*Bubble Sort Program\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <stdio.h>

int main()

{

int numarray[50], numOfEle, c, d, swap,iteration,currElement;

printf("Enter number of elements\n");

scanf("%d", &numOfEle);

printf("Enter %d integers\n", numOfEle);

for (c = 0; c < numOfEle; c++)

scanf("%d", &numarray[c]);

iteration=numOfEle;

//for (c = 0 ; c < ( numOfEle - 1 ); c++)

while(iteration>1)

{

for (currElement = 0; currElement< iteration-1; currElement++)

{

if (numarray[currElement] > numarray[currElement+1]) /\* For decreasing order use < \*/

{

swap = numarray[currElement];

numarray[currElement] = numarray[currElement+1];

numarray[currElement+1] = swap;

}

}

iteration=iteration-1;

}

printf("Sorted list in ascending order:\n");

for ( c = 0 ; c < numOfEle ; c++ )

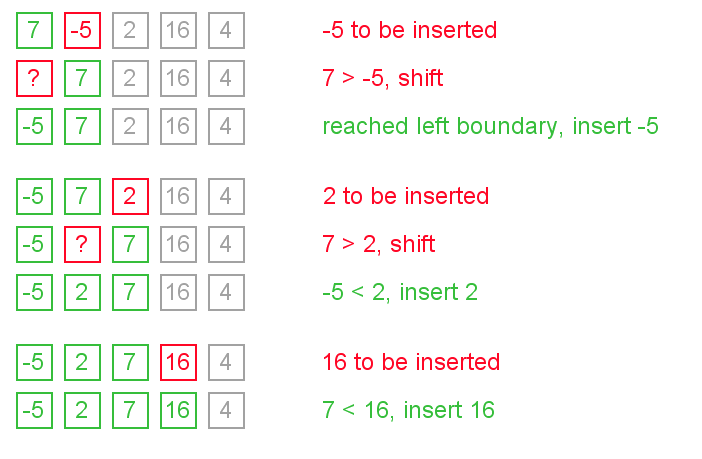
printf( " %d ", numarray[c]);

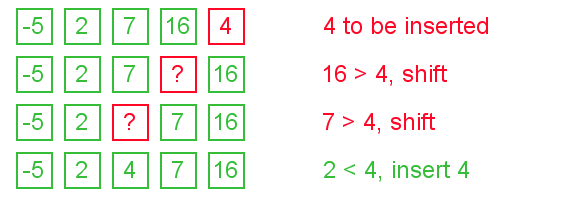
getch();

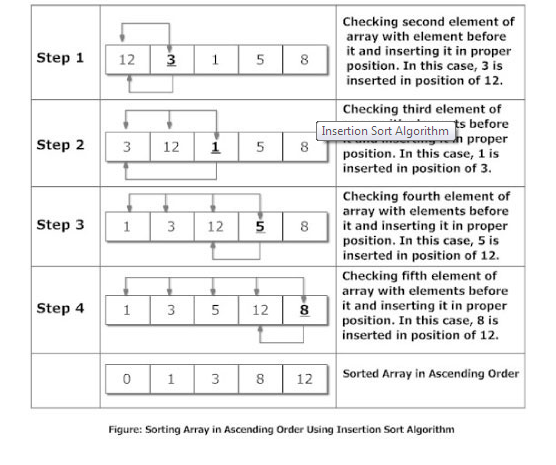
return 0;

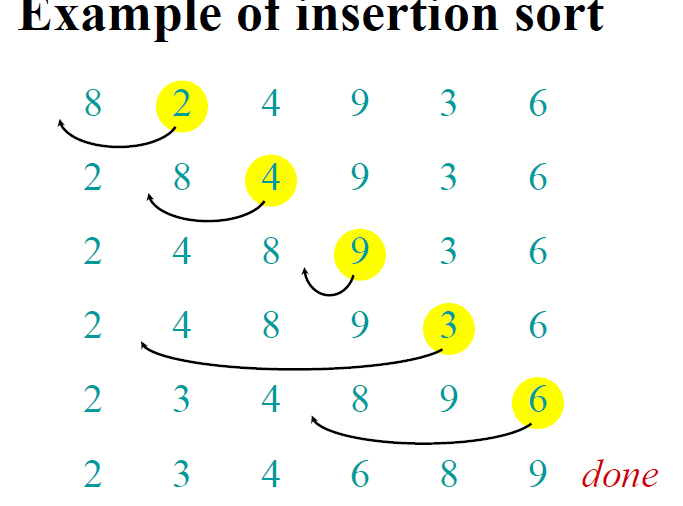
}}

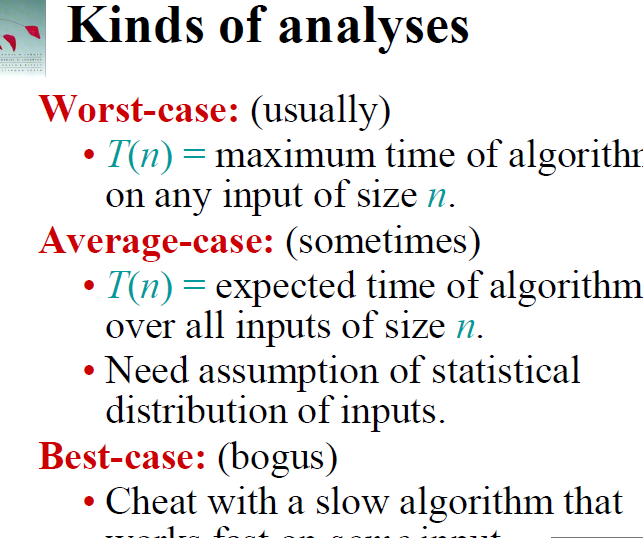
**\*\*\*\*\*\*\*\*Insertion Sort Algorithm\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

****

****

****

****

****

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Insertion Sort Program\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include<conio.h>

#include<stdio.h>

int main()

{

int noOfEle, numarray[1000], c, location, temp;

printf("Enter number of elements\n");

scanf("%d", &noOfEle);

printf("Enter %d integers\n", noOfEle);

for (c = 0; c < noOfEle; c++)

{

scanf("%d", &numarray[c]);

}

//50 29 37 5 9

//consider first is always sorted

for (c = 1 ; c <= noOfEle - 1; c++)

{

location = c;

while ( location > 0 && numarray[location] < numarray[location-1])

{

temp = numarray[location];

numarray[location] = numarray[location-1];

numarray[location-1] = temp;

location--;

}

}

printf("Sorted list in ascending order:\n");

for (c = 0; c <= noOfEle - 1; c++) {

printf(" %d ", numarray[c]);

}

getch();

return 0;

}

**\*\*\*\*\*\*Quick Sort Algorithm\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

/\*

Quicksort, or partition-exchange sort, is a sorting algorithm that,

on average, makes O(n log n) comparisons to sort n items.

It was developed by Tony Hoare. Quicksort is faster in practice than other O(n log n) algorithms such as Bubble sort or Insertion Sort. Quicksort can be implemented with an in-place partitioning algorithm, so the entire sort can be done with only O(log n) additional space.

Quicksort is a comparison sort and is not a stable sort.

Its complexity is as follows:

Best Case - O(n log n)

Worst Case - O(n^2)

Average Case - O(n log n)

Quicksort is a divide and conquer algorithm.

Quicksort first divides a large list into two smaller sub-lists:

the low elements and the high elements. Quicksort can then recursively

sort the sub-lists.

The steps are:

1. Pick an element, called a pivot, from the list.

2. Reorder the list so that all elements with values less than the pivot come

before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.

3. Recursively sort the sub-list of lesser elements and the sub-list of greater elements.

The base case of the recursion are lists of size zero or one, which never need to be sorted.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Quick Sort Program\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include"stdio.h"

int **split (** int\*, int, int **) ;**

int **main( )**

{

int **arr[10] = { 11, 2, 9, 13, 57, 25, 17, 1, 90, 3 } ;**

int **i ;**

void **quicksort (** int **\*,** int, int **) ;**

**quicksort ( arr, 0, 9 ) ;**

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

for **( i = 0 ; i <= 9 ; i++ )**

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

**getch();**

return **0;**

}

void **quicksort (** int **a[ ],** int **lower,** int **upper )**

{

int **i ;**

if **( upper > lower )**

**{**

**i = split ( a, lower, upper ) ;**

**quicksort ( a, lower, i - 1 ) ;**

**quicksort ( a, i + 1, upper ) ;**

**}**

}

int **split (** int **a[ ],** int **lower,** int **upper )**

{

int **i, p, q, t ;**

**p = lower + 1 ;**

**q = upper ;**

**i = a[lower] ;**

while **( q >= p )**

**{**

while **( a[p] < i )**

**p++ ;**

while **( a[q] > i )**

**q-- ;**

if **( q > p )**

**{**

**t = a[p] ;**

**a[p] = a[q] ;**

**a[q] = t ;**

**}**

**}**

**t = a[lower] ;**

**a[lower] = a[q] ;**

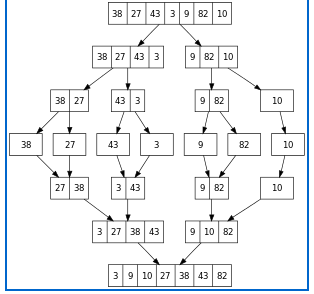
**a[q] = t ;**

return **q ;**

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*Merge Sort Algorithm\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Merge sort is very predictable. It makes between 0.5\*lg(n) and lg(n) comparisons per element, and between lg(n) and 1.5\*lg(n) swaps per element. The minima are achieved for already sorted data; the maxima are achieved, on average, for random data. If using Θ(n) extra space is of no concern, then merge sort is an excellent choice: It is simple to implement, and it is the only stable O(n·lg(n)) sorting algorithm. Note that when sorting linked lists, merge sort requires only Θ(lg(n)) extra space (for recursion).

****

**\*\*\*\*\*\*\*\*\*\*\*\*\*MergeSort Demo\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include<conio.h>

#include<stdio.h>

void **printArray(**int **\*,** int **);**

void **merge(**int **\*,** int **,** int, int **);**

void **mergeSort(**int **\*,** int **l,** int **r);**

/\* Driver program to test above functions \*/

int **main()**

{

int **arr[] = {12, 11, 13, 5, 6, 7};**

//int arr\_size = sizeof(arr)/sizeof(arr[0]);

int **arr\_size = 6;**

**printf(**"Given array is \n");

**printArray(arr, arr\_size);**

**mergeSort(arr, 0, arr\_size - 1);**

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

**printArray(arr, arr\_size);**

**getch();**

return **0;**

}

/\* Function to merge the two haves

arr[l..m] and arr[m+1..r] of array arr[]

\*/

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[20], R[20];**

/\* 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;**

**j = 0;**

**k = l;**

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)**

**{**

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

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

//overflow for large l and h

**mergeSort(arr, l, m);**

**mergeSort(arr, m+1, r);**

**merge(arr, l, m, r);**

**}**

}

/\* UITLITY FUNCTIONS \*/

/\* Function to print an array \*/

void **printArray(**int **A[],** int **size)**

{

int **i;**

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

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

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

}

**\*\*\*\*\*\*\*Normal Queue\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include<conio.h>

#include<stdio.h>

#define N 6

int queue[N]={0};

int rear=0,front=0;

void insert(void);

void del(void);

void disp(void);

void cre(void);

void main()

{

int user=0;

while(user!=4)

{

// clrscr();

printf("\n\n\n\t\t\t THE SIZE OF QUEUE IS %d",N);

printf("\n\t 1.INSERT");

printf("\n\t 2.DELETE");

printf("\n\t 3.DISPLAY");

printf("\n\t 4.EXIT");

printf("\n\t 5.CREATE \n");

scanf("%d",&user);

switch(user)

{

case 1:

insert();

break;

case 2:

del();

break;

case 3:

disp();

break;

case 4:

printf("\n\t THANK U");

break;

case 5:

cre();

break;

}

getch();

}

getch();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*insert\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void insert(void)

{

int t;

if(rear<N)

{

printf("\n\t ENTER A VALUE IN QUEUE: ");

scanf("%d",&t);

queue[rear]=t;

rear++;

}

else

{

printf("\n\t Q OVERFLOW!!!!!!!!!!!!!!!");

}

}

void del(void)

{

int i;

printf("\n\t %d gets deleted.........:",queue[front]);

queue[front]=0;

front++;

}

void disp(void)

{

int i;

for(i=front;i<rear;i++)

{

printf("\t %d",queue[i]);

}

}

void cre(void)

{

int t;

printf("\n\t ENTER A VALUE IN QUEUE :");

scanf("%d",&t);

front=0;

queue[front]=t;

rear=front+1;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Circular Queue\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

/\*\*\*\* Program to Implement Circular Queue using Array \*\*\*\*/

#include<stdio.h>

#define SIZE 5

void insert();

void delet();

void display();

int queue[SIZE], rear=-1, front=-1, item;

main()

{

int ch;

do

{

printf("\n\n1.\tInsert\n2.\tDelete\n3.\tDisplay\n4.\tExit\n");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch(ch)

{

case 1:

insert();

break;

case 2:

delet();

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("\n\nInvalid choice. Pleasr try again...\n");

}

} while(1);

getch();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void insert()

{

if((front==0 && rear==SIZE-1) || (front==rear+1))

printf("\n\nQueue is full.");

else

{

printf("\n\nEnter ITEM: ");

scanf("%d", &item);

if(rear == -1)

{

rear = 0;

front = 0;

}

else if(rear == SIZE-1)

rear = 0;

else

rear++;

queue[rear] = item;

printf("\n\nItem inserted: %d\n", item);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void delet()

{

if(front == -1)

printf("\n\nQueue is empty.\n");

else

{

item = queue[front];

if(front == rear)

{

front = -1;

rear = -1;

}

else if(front == SIZE-1)

front = 0;

else

front++;

printf("\n\nITEM deleted: %d", item);

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void display()

{

int i;

if((front == -1) || (front==rear+1))

printf("\n\nQueue is empty.\n");

else

{

printf("\n\n");

for(i=front; i<=rear; i++)

printf("\t%d",queue[i]);

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Search For Sorted Element\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <stdio.h>

#include <conio.h>

int main()

{

int c, first, last, middle, number, search, numArray[100];

printf("\nEnter number of elements\n");

scanf("%d",&number);

printf("\nEnter %d Numbers\n",number);

for ( c = 0 ; c < number ; c++ )

scanf("%d",&numArray[c]);

printf("\n \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UR Array Is \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

for ( c = 0 ; c < number ; c++ )

printf(" %d ",numArray[c]);

printf("\nEnter value to find\n");

scanf("%d",&search);

printf(" U Have To search %d ",search);

first = 0;

printf(" First : %d ",first);

last = number- 1;

printf(" Last : %d ",last);

middle = (first+last)/2;

printf(" Middle : %d ",middle);//middle=4

//search 4

while(first<=last)// 7 8 9 10 45 67 89 90 100// search 4 67 first=0 last=8 middle=4 midele=

{

if ( numArray[middle] < search )//67

{

first = middle + 1;//5

}

else if ( numArray[middle] == search )

{

printf("\n %d Found at location %d.\n", search, middle+1);

break;

}

else

{

last = middle - 1;//last=3 for 4

}

middle = (first + last)/2; //for 4 middle =0+3/2=1 means 8 is middle// for 67 middle=7 //89 is middle

}

if ( first > last )

printf(" \n Not found! %d is not present in the list.\n", search);

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Linear Search\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include <stdio.h>

#include <conio.h>

int main()

{

int myarray[100], search, c, n;

printf("Enter the number of elements in array\n");

scanf("%d",&n);

printf("Enter %d integer(s)\n", n);

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

scanf("%d", &myarray[c]);

printf("Enter the number to search\n");

scanf("%d", &search);

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

{

if (myarray[c] == search) /\* if required element found \*/

{

printf("%d is present at location %d.\n", search, c+1);

break;

}

}

getch();

return 0;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Stack\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#define MAX\_SIZE 5

int stack[MAX\_SIZE];

void push();

int pop();

void traverse();

int is\_empty();

int top\_element();

int top = -1;

main()

{

int element, choice;

while(1)

{

printf("Stack Operations.\n");

printf("1. Insert into stack (Push operation).\n");

printf("2. Delete from stack (Pop operation).\n");

printf("3. Print top element of stack.\n");

printf("4. Check if stack is empty.\n");

printf("5. Traverse stack.\n");

printf("6. Exit.\n");

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

scanf("%d",&choice);

switch ( choice )

{

case 1:

if ( top == MAX\_SIZE - 1 )

printf("Error: Overflow\n\n");

else

{

printf("Enter the value to insert.\n");

scanf("%d",&element);

push(element);

}

break;

case 2:

if ( top == -1 )

printf("Error: Underflow.\n\n");

else

{

element = pop();

printf("Element removed from stack is %d.\n", element);

}

break;

case 3:

if(!is\_empty())

{

element = top\_element();

printf("Element at the top of stack is %d\n\n", element);

}

else

printf("Stack is empty.\n\n");

break;

case 4:

if(is\_empty())

printf("Stack is empty.\n\n");

else

printf("Stack is not empty.\n\n");

break;

case 5:

traverse();

break;

case 6:

exit(0);

}

}

getch();

return 0;

}

void push(int value)

{

top++;

stack[top] = value;

}

int pop()

{

int element;

if ( top == -1 )

return top;

element = stack[top];

top--;

return element;

}

void traverse()

{

int d;

if ( top == - 1 )

{

printf("Stack is empty.\n\n");

return;

}

printf("There are %d elements in stack.\n", top+1);

for ( d = top ; d >= 0 ; d-- )

printf(" %d ", stack[d]);

printf("\n");

}

int is\_empty()

{

if ( top == - 1 )

return 1;

else

return 0;

}

int top\_element()

{

return stack[top];

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Binary Tree\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

# include<stdio.h>

# include<conio.h>

# include<stdlib.h>

typedefstruct **BST**

{

int **data;**

struct **BST \*lchild,\*rchild;**

}node;

void **insert(node \*,node \*);**

void **inorder(node \*);**

void **preorder(node \*);**

void **postorder(node \*);**

node \*search(node \*,int,node \*\*);

void **main()**

{

int **choice;**

char **ans=**'N';

int **key;**

**node \*new\_node,\*root,\*tmp,\*parent;**

**node \*get\_node();**

**root=NULL;**

**printf(**"nProgram For Binary Search Tree ");

do

**{**

**printf(**"\n1.Create");

**printf(**"\n2.Search");

**printf(**"\n3.Recursive Traversals");

**printf(**"\n4.Exit");

**printf(**"\nEnter your choice :");

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

switch(choice)

**{**

case **1:**

do

**{**

**new\_node=get\_node();**

**printf(**"\nEnter The Element ");

**scanf(**"%d",&new\_node->data);

if(root==NULL) /\* Tree is not Created \*/

**root=new\_node;**

else

**insert(root,new\_node);**

**printf(**"\nWant To enter More Elements?(y/n)");

**ans=getch();**

**}**while(ans=='y');

break;

case **2:**

**printf(**"\nEnter Element to be searched :");

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

**tmp = search(root,key,&parent);**

**printf(**"\nParent of node %d is %d",

**tmp->data,parent->data);**

break;

case **3:**

if(root==NULL)

**printf(**"\nTree Is Not Created");

else

**{**

**printf(**"\nThe Inorder display : ");

**inorder(root);**

**printf(**"\nThe Preorder display : ");

**preorder(root);**

**printf(**"\nThe Postorder display : ");

**postorder(root);**

**}**

break;

**}**

**}**while(choice!=4);

**getch();**

return **0;**

}

/\*

Get new Node

\*/

node \*get\_node()

**{**

**node \*temp;**

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

**temp->lchild=NULL;**

**temp->rchild=NULL;**

return **temp;**

**}**

/\*

This function is for creating a binary search tree

\*/

void **insert(node \*root,node \*new\_node)**

{

if(new\_node->data < root->data)

**{**

if(root->lchild==NULL)

**root->lchild = new\_node;**

else

**insert(root->lchild,new\_node);**

**}**

if(new\_node->data > root->data)

**{**

if(root->rchild==NULL)

**root->rchild=new\_node;**

else

**insert(root->rchild,new\_node);**

**}**

}

/\*

This function is for searching the node from

binary Search Tree

\*/

node \*search(node \*root,int **key,node \*\*parent)**

{

**node \*temp;**

**temp=root;**

while(temp!=NULL)

**{**

if(temp->data==key)

**{**

**printf(**"\n The %d Element is Present",temp->data);

return **temp;**

**}**

**\*parent=temp;**

if(temp->data>key)

**temp=temp->lchild;**

else

**temp=temp->rchild;**

**}**

return **NULL;**

}

/\*

This function displays the tree in inorder fashion

\*/

void **inorder(node \*temp)**

{

if(temp!=NULL)

**{**

**inorder(temp->lchild);**

**printf(**" %d ",temp->data);

**inorder(temp->rchild);**

**}**

}

/\*

This function displays the tree in preorder fashion

\*/

void **preorder(node \*temp)**

{

if(temp!=NULL)

**{**

**printf(**" %d ",temp->data);

**preorder(temp->lchild);**

**preorder(temp->rchild);**

**}**

}

/\*

This function displays the tree in postorder fashion

\*/

void **postorder(node \*temp)**

{

if(temp!=NULL)

**{**

**postorder(temp->lchild);**

**postorder(temp->rchild);**

**printf(**" %d ",temp->data);

**}**

}