1.. Given a set of numbers like <10, 36, 54,89,12> we want to find sum of weights based on the following conditions  
    1. 5 if a perfect square  
    2. 4 if multiple of 4 and divisible by 6  
    3. 3 if even number

And sort the numbers based on the weight and print it as follows

<10,its\_weight>,<36,its weight><89,its weight>

Should display the numbers based on increasing order.

2 Given a 9×9 sudoku we have to evaluate it for its correctness. We have to check both the sub matrix correctness and the whole sudoku correctness.

[Sudoku Correctness Logic](http://markandayannotes.blogspot.com/2015/08/sudoku-correctness-logic.html)

**Given a 9×9 sudoku we have to evaluate it for its correctness. We have to check both the sub matrix correctness and the whole sudoku correctness.**

bool CheckSudoku(**int**[,] sudoku)

{

**int** flag = 0;

// Check rows

**for**(**int** row = 0; row < 9; row++)

{

flag = 0;

**for** (**int** col = 0; col < 9; col++)

{

// edited : check range step (see comments)

**if** ((sudoku[row, col] < 1)||(sudoku[row, col] > 9))

{

**return** false;

}

// if n-th bit is set.. but you can use a bool array for readability

**if** ((flag & (1 << sudoku[row, col])) != 0)

{

**return** false;

}

// set the n-th bit

flag |= (1 << sudoku[row, col]);

}

}

// Check columns

**for**(**int** col= 0; col < 9; col++)

{

flag = 0;

**for** (**int** row = 0; row < 9; row++)

{

**if** ((flag & (1 << sudoku[row, col])) != 0)

{

**return** false;

}

flag |= (1 << sudoku[row, col]);

}

}

// Check 3x3 boxes

**for**(**int** box= 0; box < 9; box++)

{

flag = 0;

**for** (**int** ofs = 0; ofs < 9; ofs++)

{

**int** col = (box % 3) \* 3;

**int** row = ((**int**)(box / 3)) \* 3;

**if** ((flag & (1 << sudoku[row, col])) != 0)

{

**return** false;

}

flag |= (1 << sudoku[row, col]);

}

}

**return** true;

#### 3.Given a two dimensional array of string like

#### <”luke”, “shaw”>

#### <”wayne”, “rooney”>

#### <”rooney”, “ronaldo”>

#### <”shaw”, “rooney”>

#### Where the first string is “child”, second string is “Father”. And given “ronaldo” we have to find his no of grandchildren Here “ronaldo” has 2 grandchildren.

#### 

#### So our output should be 2..

#### 

#### My solution ::

#include <iostream>

#include<cstdlib>

**using** namespace std;

**int** count=0;

string a[4][2]={

{"luke", "shaw"},

{"wayne", "rooney"},

{"rooney", "ronaldo"},

{"shaw", "rooney"}

};

**void** grandchi(string x){

**for**(**int** i=0;i<4;i++){

**if**(x==a[i][1]){

count++;

}

}

}

**int** **main**() {

string s,s1;

**for**(**int** i=0;i<4;i++){

**for**(**int** j=0;j<2;j++){

cout<<a[i][j]<<"\n";

}

}

cout<<"enter string"<<"\n";

cin>>s;

cout<<s;

**for**(**int** i=0;i<4;i++){

**if**(s==a[i][1]){

s1=a[i][0];

grandchi(s1);

}

}

cout<<"grand children:"<<count;

**return** 0;

}

4.Find the sum of the digits of given number.

Eg:

Input:

14->1+4->5.

165->1+6+5->1+2->3

2.Find the occurrence of character in given string. Should not use additional memory.

Eg: AAAABCCCDD

o/p: A4BC3D2

ignore the character that is repeated again.

Eg 1: Input: a1b10

Output: abbbbbbbbbb

Eg: 2: Input: b3c6d15

Output: bbbccccccddddddddddddddd

The number varies from 1 to 99.

**MY SOLUTION ::**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

**int** **main**() {

**char** s[10];

**int** i = 0;

**int** count,j;

**int** inc;

scanf("%s",s);

**while** (1==sscanf(s+i, "%\*[^0-9]%d%n", &count, &inc))

{

**for**(j=0;j<count;j++)

printf("%c",\*(s+i));

i += inc;

count = 1;

}

**return** 0;

}

5.Finding the kth smallest element in an array without sorting the elements in the array.

#include<stdio.h>

#include<conio.h>

int a[15]={1,4,5,3,12,45,63,32,134},n=9,k,temp[15],num;

void kth\_largest()

{

int i,j;

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

temp[i]=a[i];

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

{

for(j=0;j<n-1;j++)

{

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

temp[j]=temp[j]+temp[j+1]-(temp[j+1]=temp[j]);

}

}

printf("\n%d^th largest element= %d\n\n",k,temp[n-k]);

}

void kth\_smallest()

{

int i,j;

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

temp[i]=a[i];

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

{

for(j=0;j<n-1;j++)

{

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

temp[j]=temp[j]+temp[j+1]-(temp[j+1]=temp[j]);

}

}

printf("\n%d^th smallest element= %d\n\n",k,temp[n-k]);

}

int main()

{

printf("\nEnter the value of k: ");

scanf("%d",&k);

int i;

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

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

printf("\n\n");

kth\_largest();

kth\_smallest();

getch();

return 0;

}

1. Spiral matrix  
   Given a square matrix(n is odd) , print the elements in spiral order starting from the center till a[0][0]  
   A=  
   1 2 3  
   4 5 6  
   7 8 9  
   o/p 5 4 7 8 9 6 3 2 1

#include<stdio.h>

#include<conio.h>

#define left 1

#define down 2

#define right 3

#define up 4

int matrix[5][5]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15},{16,17,18,19,20,21},{22,23,24,25,26}};

int i,j,direction=1,order=5,final\_left,final\_right,final\_up,final\_down;

void spiral()

{

while(i!=final\_up-1 || j!=final\_left-1)

{

switch(direction)

{

case left:

while(j>=final\_left-1)

printf("%d ",matrix[i][j--]);

j++;

if(i==final\_up-1 && j==final\_left-1)

break;

i++;

direction=down;

break;

case down:

while(i<=final\_down+1)

printf("%d ",matrix[i++][j]);

i--;

j++;

direction=right;

break;

case right:

while(j<=final\_right+1)

printf("%d ",matrix[i][j++]);

j--;

i--;

direction=up;

break;

case up:

while(i>=final\_up-1)

printf("%d ",matrix[i--][j]);

i++;

j--;

direction=left;

break;

}

}

}

int main()

{

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

{

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

printf("%4d",matrix[i][j]);

printf("\n");

}

printf("\n\n");

i=order/2;j=order/2;

final\_left=order/2;

final\_right=order/2;

final\_up=order/2;

final\_down=order/2;

while(final\_left>0 && final\_up>0)

{

spiral();

final\_left--;

final\_up--;

final\_right++;

final\_down++;

i=final\_up;

j=final\_left-1;

}

getch();

return 0;

}

1. Lower triangle matrix

Given “n” print the following  
N=3  
1  
6 2  
5 4 3

#include<stdio.h>

#include<conio.h>

#define filled 1

#define empty 0

int left,diag,top,startx,starty;

int a[11][11],count[11][11],counter=0;

int N,i,j,num=1;

void init()

{

for(i=0;i<=10;i++)

{

for(j=0;j<=10;j++)

{

a[i][j]=0;

count[i][j]=empty;

}

}

}

void move\_diagnol()

{

for(i=startx,j=starty;i<=diag && j<=diag ;i++,j++)

{

if(count[i][j]==filled)

continue;

a[i][j]=num;

count[i][j]=filled;

num++;

}

}

void move\_left()

{

for(j=starty;j>=left;j--)

{

if(count[i][j]==filled)

continue;

a[i][j]=num;

count[i][j]=filled;

num++;

}

}

void move\_top()

{

for(i=startx;i>=top;i--)

{

if(count[i][j]==filled)

continue;

a[i][j]=num;

count[i][j]=filled;

num++;

}

}

void begin()

{

while(counter<=3)

{

move\_diagnol();

--diag;

starty=--j;

startx=--i;

move\_left();

left++;

starty=++j;

move\_top();

top++;

++i;

startx=++i+1;

starty=++j;

counter++;

}

}

void display()

{

int i,j;

char c=" ";

for(i=1;i<=10;i++)

{

for(j=1;j<=10;j++)

{

if(a[i][j])

printf("%4d",a[i][j]);

else

printf("%4c",c);

}

printf("\n");

}

}

int main()

{

init();

printf("\nEnter the value of N: ");

scanf("%d",&N);

left=1;

top=1;

diag=N;

startx=top;

starty=top;

printf("\n\n");

begin();

display();

getch();

return 0;

}

1. Write a C program to Remove even numbers and fill it with zeros.  
   Note: You can use only one loop for your logic.  
   Two variables apart from the array.  
   Input array 1,2,3,4,5,6  
   Output 1,3,5,0,0,0

#include<stdio.h>

#include<conio.h>

#define true 1

#define false -1

/\*

1)

Input array 1,2,3,4,5,6

Output 1,3,5,0,0,0

Remove even nos and fill it with zeros

U can use only one loop for ur logic

Two variables apart from the array.

\*/

int main()

{

int a[10]={3,4,5,2,1,8,9,10};

int n=7;

int i=0,j=0,l=0,even=true,zeros=false,sort=false,startprint=false,print=false;

for(i=0; even==true || sort==false || zeros==false; )

{

//even no=0

if(even==true)

{

if(j<n)

{

if(a[j]%2==0)

a[j]=0;

j++;

}

else

{

even=false;

j=0;

}

}

//sort

if(even==false && sort==false)

{

if(j<n-1)

{

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

{

a[j]=a[j]+a[j+1]-(a[j+1]=a[j]);

j++;

}

else

j++;

}

else

{

i++;

if(i>=n)

sort=true;

j=0;

}

}

//print

if(sort==true && zeros==false)

{

if(startprint==false)

{

if(a[j]==0)

j++;

else

startprint=true;

}

else

{

if(print==false)

{

i=j;

print=true;

}

else if(print==true)

{

if(i<n)

{

a[l]=a[l]+a[i]-(a[i]=a[l]);

i++;

}

if(l<n)

{

printf("%d ",a[l++]);

}

else

zeros=true;

}

}

}

}

getch();

return 0;

}

1. Given an array , find the nth largest and nth smallest number. Sorting should not be used

#include<stdio.h>

#include<conio.h>

int a[15]={1,4,5,3,12,45,63,32,134},n=9,k,temp[15],num;

void kth\_largest()

{

int i,j;

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

temp[i]=a[i];

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

{

for(j=0;j<n-1;j++)

{

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

temp[j]=temp[j]+temp[j+1]-(temp[j+1]=temp[j]);

}

}

printf("\n%d^th largest element= %d\n\n",k,temp[n-k]);

}

void kth\_smallest()

{

int i,j;

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

temp[i]=a[i];

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

{

for(j=0;j<n-1;j++)

{

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

temp[j]=temp[j]+temp[j+1]-(temp[j+1]=temp[j]);

}

}

printf("\n%d^th smallest element= %d\n\n",k,temp[n-k]);

}

int main()

{

printf("\nEnter the value of k: ");

scanf("%d",&k);

int i;

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

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

printf("\n\n");

kth\_largest();

kth\_smallest();

getch();

return 0;

}

1. (Partitioning and merging)  
   Given an array and a partition size, you have to partition the array with that value , then we will specify the partition order , you have to merge based on that order  
   Array : 1 2 3 4 5  
   Partition size 2  
   (so the array will be partitioned as 1 2, 3 4, 5)  
   Partition order 3 2 1  
   o/p: 5 3 4 1 2

#include<stdio.h>

#include<conio.h>

int a[10]={1,3,4,2,5,6,9,12},b[10][10],size,merge[10],row=0,col=0,n=8,aptr=0;

void init()

{

int i,j;

for(i=0;i<10;i++)

for(j=0;j<10;j++)

b[i][j]=-1;

}

void partition()

{

int k=0,i;

while(k<n)

{

for(; k==0 || (k%size!=0 && k<n) ; k++)

{

b[row][col++]=a[k];

}

//if(k>n)

//b[row][col]=a[k];

//else

{

row++;

col=0;

b[row][col++]=a[k++];

}

}

for(k=0;k<row;k++)

{

printf("Partition %d \n\n",k+1);

for(i=0;i<10;i++)

{

if(b[k][i]==-1)

break;

else

printf("%d ",b[k][i]);

}

printf("\n\n");

}

}

void merger()

{

int i,j;

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

{

printf("\nEnter the order: ");

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

}

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

{

for(j=0;j<10;j++)

{

if(b[merge[i]-1][j]==-1)

break;

else

a[aptr++]=b[merge[i]-1][j];

}

}

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

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

}

int main()

{

printf("\nEnter the partition size: ");

scanf("%d",&size);

init();

partition();

merger();

getch();

return 0;

}

1. **Array partition merge order**

#include<stdio.h>

#include<conio.h>

int a[10]={1,3,4,2,5,6,9,12},b[10][10],size,merge[10],row=0,col=0,n=8,aptr=0;

void init()

{

int i,j;

for(i=0;i<10;i++)

for(j=0;j<10;j++)

b[i][j]=-1;

}

void partition()

{

int k=0,i;

while(k<n)

{

for(; k==0 || (k%size!=0 && k<n) ; k++)

{

b[row][col++]=a[k];

}

//if(k>n)

//b[row][col]=a[k];

//else

{

row++;

col=0;

b[row][col++]=a[k++];

}

}

for(k=0;k<row;k++)

{

printf("Partition %d \n\n",k+1);

for(i=0;i<10;i++)

{

if(b[k][i]==-1)

break;

else

printf("%d ",b[k][i]);

}

printf("\n\n");

}

}

void merger()

{

int i,j;

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

{

printf("\nEnter the order: ");

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

}

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

{

for(j=0;j<10;j++)

{

if(b[merge[i]-1][j]==-1)

break;

else

a[aptr++]=b[merge[i]-1][j];

}

}

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

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

}

int main()

{

printf("\nEnter the partition size: ");

scanf("%d",&size);

init();

partition();

merger();

getch();

return 0;

}

1. **DNA pattern**

**#include<stdio.h>**

**#include<conio.h>**

**/\*Print the following pattern given width (odd number) and height(should be multiple of width)**

**Width 7**

**Height 21**

**(dna pattern)**

**\*/**

**int main()**

**{**

**int i,j,n=21,H,h;**

**printf("\nEnter the height: ");**

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

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

**for(h=1;h<=H;h++)**

**{**

**for(i=4;i>=1;i--)**

**{**

**for(j=4;j>=1;j--)**

**{**

**if(j==i)**

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

**else**

**printf(" ");**

**}**

**for(j=1;j<=4;j++)**

**{**

**if(j==i)**

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

**else**

**printf(" ");**

**}**

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

**}**

**for(i=2;i<=4;i++)**

**{**

**for(j=4;j>=1;j--)**

**{**

**if(j==i)**

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

**else**

**printf(" ");**

**}**

**for(j=1;j<=4;j++)**

**{**

**if(j==i)**

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

**else**

**printf(" ");**

**}**

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

**}**

**}**

**getch();**

**return 0;**

**}**

1. **Weight sort**

Input:    43 3 91 95 25 32 80

Output:  80 25 95 -1 32 -1 -1

**int** **main**(){

**int** a[] = {43,3,91,95,25,32,80};

**int** i,max,j,temp,k,count;

**int** n = **sizeof**(a)/**sizeof**(a[0]);

clrscr();

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

max = 0;

count=0;

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

count++;

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

**if**(max==0)

max=a[j];

**else** **if**(max>a[j])

max = a[j];

}

}

**if**(max>a[i] && count>1)

a[i] = max;

**else**

a[i] = -1;

}

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

printf("%5d",a[i]);

getch();

}

1. You are provided with a set of keywords. For a given input whenever a keyword is encountered, mark it as “Most used”. If the keyword is not present in the given input then, replace it with the data in the input and mark it as “Least used”. When all the keywords are either “Most used” or “Least used”, start replacing the first keyword whenever a new data is encountered in the input.

Explanation:

Keyword set = [2 4 8 9]

When 2 is entered from the keyboard, the keyword 2 is marked as “Most used”. Similarly, when 8 and 9 are entered, they are marked as “Most used”.

When 7 is entered, then from the keyword set 4 is replaced with 7 and is marked as “Least used”. This process continues till all the elements are either marked as “most used” or “Least used” and from then on the replacements starts with the first number.

#include <stdio.h>

void main()

{

int i, n, a[50], k[50],count=0, key,flag=0;

/\* Array a[50] is used to store the elements. Array k[50] is used to find whether the marked as “most used” or “Least used”. If element in array k is set to 1 denotes it is marked, if not it denotes it is unmarked

Count is used to track whether all the element of array are called or not.

Flag is used for setting Most Used and Least Used.

\*/

printf("Enter the size of keyword set.");

scanf("%d",&n);

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

{

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

k[i]=0;

}

while(count != n)

{

printf("Enter a key:");

scanf("%d",&key);

printf("\n");

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

{

if(a[i] == key) // If key is present in the array.

{

k[i]=1;

printf("most used\n");

flag=1;

count++;

break;

}

}

if(flag == 0)

{

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

{

if(k[i] == 0)

{

printf("least used\n");

k[i]=1;

a[i]=key;

count++;

break;

}

}

}

flag=0;

}

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

{

printf("%d\n",a[i]);

}

}

1. In the given array, traverse from left to right replacing the first element with the next greater number present to its right. If no greater number to its right is found, then replace the remaining numbers with -1.

Sample input = [2 4 8 90 77 54 ] Sample output: [4 8 54 -1 -1 -1]

/\* The two loops used in this program is similar to bubbled sorting loop logic.\*/

#include <stdio.h>

void main()

{

int a[50],n,i,j,temp;

printf("Enter the size:");

scanf("%d",&n);

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

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

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

{

temp=a[i];

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

{

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

{

if(temp==a[i])

temp=a[j];

else if(a[j]<temp)

temp=a[j];

}

}

if(temp != a[i])

/\*if temp != a[i] means swapping is done, i.e., larger element is available to the right\*/

{

a[i]=temp;

}

else /\* swapping not done, i.e., larger element is not available to the right\*/

{

a[i]=-1;

i++;

break; /\* Comes out of the loop\*/

}

}

for(;i<n;i++) /\* for all remaining elements value is assigned to -1\*/

a[i]=-1;

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

printf("\n%d",a[i]);

}

1. Divide the given array into two arrays such that both arrays have equal averages

Sample input: [2 4 8 10 16] Sample output: [8] and [2 4 10 16]

Sample input: [2 4 8 10] Sample output: [4 8] and [2 10]

#include <stdio.h>

void main()

{

int a[50],b[50],c[50],n,i,j,l,sum=0,k;

float avg=0;

printf("Enter the total number of elements:");

scanf("%d",&n);

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

{

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

avg += a[i];

}

avg=avg/n;

/\* This loop is used for grouping of elements hence it starts with 1 and goes upto n-1\*/

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

{

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

{

/\*

The below for loop is used for grouping of elements. i.e, when i=1, this loop has sum as a single elment, i=2, this loop takes sum as for 2 elements taken as a pair. And it continues.

\*/

for(j=k;j<k+i;j++)

{

sum=sum+a[j];

}

if(avg == sum/i)

{

printf("\n");

printf("Array 1\n");

for(l=k;l<k+i;l++)

{

printf("%d\t",a[l]);

}

printf("\n");

printf("Array 2\n");

for(l=0;l<k;l++)

{

printf("%d\t",a[l]);

}

for(l=k+i;l<n;l++)

{

printf("%d\t",a[l]);

}

return;

/\*return works same as exit. Since we have found the group, the program is being terminated. Since the return type of main is given void, simply return; is given\*/

}

sum=0;

}

}

/\* The program comes to this region only if the equal average arrays are not possible\*/

printf("\nIt is not possible to split the given array.");

}

1. Write a program to eliminate the reoccurrence of numbers such as changing 2233555697 as 235697 using any language such as C, JAVA etc. (3 marks)
2. Write a program to find permutation & combination using recursion using any language such as C, JAVA etc. (3 marks)

# include<stdio.h>

# include<conio.h>

#include<string.h>

void swap (char \*x, char \*y)

{

    char temp;

    temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void permute(char \*a, int i, int n)

{

    int j;

    if (i == n)

    printf("%s\n", a);

    else

    {

        for (j = i; j <= n; j++)

        {

            swap((a+i), (a+j));

            permute(a, i+1, n);

            swap((a+i), (a+j));

        }

    }

}

int main()

{

    char str[80] ;

    int len=0,i;

    puts("Enter a string:\n\n");

    gets(str);

    for (i=0; str[i] != '\0'; i++)

    {

        len++;

    }

    permute(str, 0, len);

    getchar();

    return 0;

}

// C program to print all permutations with duplicates allowed

#include <stdio.h>

#include <string.h>

/\* Function to swap values at two pointers \*/

void swap(char \*x, char \*y)

{

char temp;

temp = \*x;

\*x = \*y;

\*y = temp;

}

/\* Function to print permutations of string

This function takes three parameters:

1. String

2. Starting index of the string

3. Ending index of the string. \*/

void permute(char \*a, int l, int r)

{

int i;

if (l == r)

printf("%s\n", a);

else

{

for (i = l; i <= r; i++)

{

swap((a+l), (a+i));

permute(a, l+1, r);

swap((a+l), (a+i)); //backtrack

}

}

}

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

int main()

{

char str[] = "ABC";

int n = strlen(str);

permute(str, 0, n-1);

return 0;

}

Output:

ABC

ACB

BAC

BCA

CBA

CAB

1. Write a program to reverse the string such that ONE TWO THREE as ENO OWT EERHT without using inbuilt functions using any language such as C, JAVA. **#include**<**stdio.h**>  
   **#include**<**conio.h**>  
   **#include**<**string.h**>  
   int main()  
   {  
    char str[100];  
    int i,temp;  
    printf("Enter any string : ");  
    gets(str);  
    for(i=0; str[i]!=NULL; i++)  
    {  
     if(str[i+1]==' ' **||** str[i+1]==NULL)  
     {  
      for(temp=i; temp>=0 **&&** str[temp]!=' '; temp--)  
        printf("%c", str[temp]);  
     }  
     printf(" ");  
    }  
    getch();  
    **return 0**;  
   } (6 marks)
2. Write a program to print the value of the number such that 13876 as Thirteen thousand & Eight Hundred & Seventy Six using any language such as C, JAVA. The number can range from 0 to 99999 (12 marks)

#include<stdio.h>

#include<conio.h>

#include<math.h>

void dthou(int );

void dthou1(int );

void thou(int );

void hund(int );

void tens(int );

void tens1(int );

void ones(int );

void main()

{

int a,b,c,d,e;

long n;

clrscr();

printf("Enter the number: (Max. 5 digits) \n");

scanf("%ld",&n);

if(n<=99999)

{

a=n%10;

n=n/10;

b=n%10;

n=n/10;

c=n%10;

n=n/10;

d=n%10;

n=n/10;

e=n%10;

if(e==1)

{

dthou1(d);

}

else

{

dthou(e);

thou(d);

}

hund(c);

if(b!=1)

{

tens(b);

ones(a);

}

if(b==1)

{

tens1(a);

}

}

else

{

printf("Number limit exceeded\n");

}

getch();

}

void dthou(int x)

{

if(x==2)

printf("Twenty ");

if(x==3)

printf("Thirty ");

if(x==4)

printf("Forty ");

if(x==5)

printf("Fifty ");

if(x==6)

printf("Sixty ");

if(x==7)

printf("Seventy ");

if(x==8)

printf("Eighty ");

if(x==9)

printf("Ninety ");

}

void dthou1(int x)

{

if(x==0)

printf("Ten Thousand ");

if(x==1)

printf("Eleven Thousand ");

if(x==2)

printf("Twelve Thousand ");

if(x==3)

printf("Thirteen Thousand ");

if(x==4)

printf("Forteen Thousand ");

if(x==5)

printf("Fivteen Thousand ");

if(x==6)

printf("Sixteen Thousand ");

if(x==7)

printf("Seventeen Thousand ");

if(x==8)

printf("Eighteen Thousand ");

if(x==9)

printf("Nineteen Thousand ");

}

void thou(int x)

{

if(x!=0)

{

if(x==1)

printf("One Thousand ");

if(x==2)

printf("Two Thousand ");

if(x==3)

printf("Three Thousand ");

if(x==4)

printf("Four Thousand ");

if(x==5)

printf("Five Thousand ");

if(x==6)

printf("Six Thousand ");

if(x==7)

printf("Seven Thousand ");

if(x==8)

printf("Eight Thousand ");

if(x==9)

printf("Nine Thousand ");

}

}

void hund(int x)

{

if(x!=0)

{

if(x==1)

printf("One Hundred ");

if(x==2)

printf("Two Hundred ");

if(x==3)

printf("Three Hundred ");

if(x==4)

printf("Four Hundred ");

if(x==5)

printf("Five Hundred ");

if(x==6)

printf("Six Hundred ");

if(x==7)

printf("Seven Hundred ");

if(x==8)

printf("Eight Hundred ");

if(x==9)

printf("Nine Hundred ");

}

}

void tens(int x)

{

if(x!=0)

{

if(x==2)

printf("Twenty ");

if(x==3)

printf("Thirty ");

if(x==4)

printf("Forty ");

if(x==5)

printf("Fifty ");

if(x==6)

printf("Sixty ");

if(x==7)

printf("Seventy ");

if(x==8)

printf("Eighty ");

if(x==9)

printf("Ninety ");

}

}

void tens1(int x)

{

if(x==0)

printf("Ten\n");

if(x==1)

printf("Eleven\n");

if(x==2)

printf("Twelve\n");

if(x==3)

printf("Thirteen\n");

if(x==4)

printf("Forteen\n");

if(x==5)

printf("Fivteen\n");

if(x==6)

printf("Sixteen\n");

if(x==7)

printf("Seventeen\n");

if(x==8)

printf("Eighteen\n");

if(x==9)

printf("Nineteen\n");

}

void ones(int x)

{

if(x!=0)

{

if(x==1)

printf("One\n");

if(x==2)

printf("Two\n");

if(x==3)

printf("Three\n");

if(x==4)

printf("Four\n");

if(x==5)

printf("Five\n");

if(x==6)

printf("Six\n");

if(x==7)

printf("Seven\n");

if(x==8)

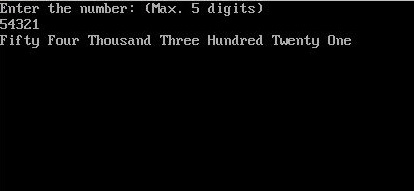
printf("Eight\n");

if(x==9)

printf("Nine\n");

}

}



1. Finding the nth element from the last in a singly linked list.

/ Simple C program to find n'th node from end

#include<stdio.h>

#include<stdlib.h>

/\* Link list node \*/

struct node

{

  int data;

  struct node\* next;

};

/\* Function to get the nth node from the last of a linked list\*/

void printNthFromLast(struct node\* head, int n)

{

    int len = 0, i;

    struct node \*temp = head;

    // 1) count the number of nodes in Linked List

    while (temp != NULL)

    {

        temp = temp->next;

        len++;

    }

    // check if value of n is not more than length of the linked list

    if (len < n)

      return;

    temp = head;

    // 2) get the (n-len+1)th node from the begining

    for (i = 1; i < len-n+1; i++)

       temp = temp->next;

    printf ("%d", temp->data);

    return;

}

void push(struct node\*\* head\_ref, int new\_data)

{

  /\* allocate node \*/

  struct node\* new\_node =

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

  /\* put in the data  \*/

  new\_node->data  = new\_data;

  /\* link the old list off the new node \*/

  new\_node->next = (\*head\_ref);

  /\* move the head to point to the new node \*/

  (\*head\_ref)    = new\_node;

}

/\* Drier program to test above function\*/

int main()

{

  /\* Start with the empty list \*/

  struct node\* head = NULL;

  // create linked 35->15->4->20

  push(&head, 20);

  push(&head, 4);

  push(&head, 15);

  push(&head, 35);

  printNthFromLast(head, 5);

  return 0;

}

1. Finding the element occurring odd number of time in an array. No extra space and O(n) time complexity.

Eg: {1,1,3,4,4,4,4}

Output : 3

Given an integer array, one element occurs odd number of times and all others have even occurrences. Find the element with odd occurrences.

This question is very similar to the previous [find even occurring element](http://www.ardendertat.com/2011/11/29/programming-interview-questions-18-find-even-occurring-element/) problem. And we can actually use the same solutions. One approach is again to build a hashtable of element occurrence counts and return the element with odd count. Both time and space complexity of this solution is O(N).

But we can do much better by using the XOR trick described in that post. It’s the following: if we XOR a number with itself odd number of times the result is 0, otherwise if we XOR even number of times the result is the number itself. So, if we XOR all the elements in the array, the result is the odd occurring element itself. Because all even occurring elements will be XORed with themselves odd number of times, producing 0. And the only odd occurring element will be XORed with itself even number of times, producing its own value.

Let’s say we’re given the following array: [1, 2, 3, 1, 2, 3, 1]. If  we XOR all the elements in this array the result is 1^2^3^1^2^3^1 = 1. Because the numbers 2 and 3 will be XORed with themselves 1 time, producing 0. And the number 1 will be XORed with itself 2 times, resulting in its own value. So, the overall result of the XOR operations is the number 1, odd occurring element in the array. Here’s the code:

|  |
| --- |
| **def** getOdd(arr):  **return** reduce(**lambda** x, y: x^y, arr) |

Simple as that! The time complexity of this solution is still O(N), but now the space complexity is constant O(1). Because we’re just using constant extra memory, not proportional to the size of the input array. This is the most optimal solution to the problem, since we’re accessing every element only once and using constant extra space.

This is a great question because it demonstrates the power and effectiveness of bit manipulation operators.

A **Simple Solution** is to run two nested loops. The outer loop picks all elements one by one and inner loop counts number of occurrences of the element picked by outer loop. Time complexity of this solution is O(n2).

A **Better Solutio**n is to use Hashing. Use array elements as key and their counts as value. Create an empty hash table. One by one traverse the given array elements and store counts. Time complexity of this solution is O(n). But it requires extra space for hashing.

The **Best Solution** is to do bitwise XOR of all the elements. XOR of all elements gives us odd occurring element. Please note that XOR of two elements is 0 if both elements are same and XOR of a number x with 0 is x.

Below are implementations of this best approach.

**Program:**

* C/C++
* Python

|  |
| --- |
| #include <stdio.h>    int getOddOccurrence(int ar[], int ar\_size)  {       int i;       int res = 0;       for (i=0; i < ar\_size; i++)          res = res ^ ar[i];         return res;  }    /\* Diver function to test above function \*/  int main()  {       int ar[] = {2, 3, 5, 4, 5, 2, 4, 3, 5, 2, 4, 4, 2};       int n = sizeof(ar)/sizeof(ar[0]);       printf("%d", getOddOccurrence(ar, n));       return 0;  } |

Run on IDE

Output:

5

I tried implementing it using a HashMap<Integer, Integer>, by mapping each integer element to a number that would identify if that number occurs even number of times. I used the XORlogic for that. 5 ^ 5 = 0, and 5 ^ anyNumberOtherThan5 != 0.

Here's my code:

public static void findEvenOccurringNum(int[] arr) {

Map<Integer, Integer> map = new HashMap<Integer, Integer>();

for (int num: arr) {

if (map.containsKey(num)) {

map.put(num, map.get(num) ^ 1);

} else {

map.put(num, 1);

}

}

for (Entry<Integer, Integer> entry: map.entrySet()) {

if (entry.getValue() == 0) {

System.out.println(entry.getKey());

}

}

}

What I don't like about this solution is the 2nd iteration. That would iterate over all the unique numbers in the array. Can there be any possible improvement here? May be a solution that avoids using a map? Or is it the best I can do?

1. A set of activities will be given along with their starting time and ending time. Write a code to find the maximum number of activities that can be performed without overlapping. Also find all possible activity sets that can be performed.

Eg:

Act1 st: 0 et: 4

Act2 st: 5 et: 6

Act3: st:0 et: 7

Act4: st:6 et: 7

Output: {1,2,4} and {3} – sets

# Greedy Algorithms | Set 1 (Activity Selection Problem)

Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. Greedy algorithms are used for optimization problems. An optimization problem can be solved using Greedy if the problem has the following property:At every step, we can make a choice that looks best at the moment, and we get the optimal solution of the complete problem.  
If a Greedy Algorithm can solve a problem, then it generally becomes the best method to solve that problem as the Greedy algorithms are in general more efficient than other techniques like Dynamic Programming. But Greedy algorithms cannot always be applied. For example, Fractional Knapsack problem (See [this](http://www.cs.binghamton.edu/~dima/cs333/knapsack.ppt)) can be solved using Greedy, but [0-1 Knapsack](http://www.geeksforgeeks.org/archives/18430) cannot be solved using Greedy.

Following are some standard algorithms that are Greedy algorithms.  
**1)**[**Kruskal’s Minimum Spanning Tree (MST)**](http://en.wikipedia.org/wiki/Kruskal%27s_algorithm)**:** In Kruskal’s algorithm, we create a MST by picking edges one by one. The Greedy Choice is to pick the smallest weight edge that doesn’t cause a cycle in the MST constructed so far.  
**2)**[**Prim’s Minimum Spanning Tree**](http://en.wikipedia.org/wiki/Prim%27s_algorithm)**:** In Prim’s algorithm also, we create a MST by picking edges one by one. We maintain two sets: set of the vertices already included in MST and the set of the vertices not yet included. The Greedy Choice is to pick the smallest weight edge that connects the two sets.  
**3)**[**Dijkstra’s Shortest Path**](http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)**:**The Dijkstra’s algorithm is very similar to Prim’s algorithm. The shortest path tree is built up, edge by edge. We maintain two sets: set of the vertices already included in the tree and the set of the vertices not yet included. The Greedy Choice is to pick the edge that connects the two sets and is on the smallest weight path from source to the set that contains not yet included vertices.  
**4)**[**Huffman Coding**](http://en.wikipedia.org/wiki/Huffman_coding)**:** Huffman Coding is a loss-less compression technique. It assigns variable length bit codes to different characters. The Greedy Choice is to assign least bit length code to the most frequent character.

The greedy algorithms are sometimes also used to get an approximation for Hard optimization problems. For example, [Traveling Salesman Problem](http://en.wikipedia.org/wiki/Travelling_salesman_problem) is a NP Hard problem. A Greedy choice for this problem is to pick the nearest unvisited city from the current city at every step. This solutions doesn’t always produce the best optimal solution, but can be used to get an approximate optimal solution.

Let us consider the [Activity Selection problem](http://en.wikipedia.org/wiki/Activity_selection_problem) as our first example of Greedy algorithms. Following is the problem statement.  
You are given n activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.  
Example:

Consider the following 6 activities.

start[] = {1, 3, 0, 5, 8, 5};

finish[] = {2, 4, 6, 7, 9, 9};

The maximum set of activities that can be executed

by a single person is {0, 1, 3, 4}

## We strongly recommend that you click here and practice it, before moving on to the solution.

The greedy choice is to always pick the next activity whose finish time is least among the remaining activities and the start time is more than or equal to the finish time of previously selected activity. We can sort the activities according to their finishing time so that we always consider the next activity as minimum finishing time activity.

1) Sort the activities according to their finishing time  
2) Select the first activity from the sorted array and print it.  
3) Do following for remaining activities in the sorted array.  
…….a) If the start time of this activity is greater than the finish time of previously selected activity then select this activity and print it.

In the following C implementation, it is assumed that the activities are already sorted according to their finish time.

* C++
* Java
* Python

|  |
| --- |
| #include<stdio.h>    // Prints a maximum set of activities that can be done by a single  // person, one at a time.  //  n   -->  Total number of activities  //  s[] -->  An array that contains start time of all activities  //  f[] -->  An array that contains finish time of all activities  void printMaxActivities(int s[], int f[], int n)  {      int i, j;        printf ("Following activities are selected \n");        // The first activity always gets selected      i = 0;      printf("%d ", i);        // Consider rest of the activities      for (j = 1; j < n; j++)      {        // If this activity has start time greater than or        // equal to the finish time of previously selected        // activity, then select it        if (s[j] >= f[i])        {            printf ("%d ", j);            i = j;        }      }  }    // driver program to test above function  int main()  {      int s[] =  {1, 3, 0, 5, 8, 5};      int f[] =  {2, 4, 6, 7, 9, 9};      int n = sizeof(s)/sizeof(s[0]);      printMaxActivities(s, f, n);      getchar();      return 0;  } |

Run on IDE

Output:

Following activities are selected

0 1 3 4

1. Given a String “velammalcollege”, write a program to print what are all the characters present in the string and find its no. of occurrence(s) in that string in the alphabetical order. i.e. Output must be A-2 C-1 E-3 G-1 L-4 M-2 O-1 V-1

void main()

{

int i,j; int c[26],count=0; char a[]="shahid";

clrscr();

for(i=0;i<26;i++)

{

count=0;

for(j=0;j<strlen(a);j++)

{

if(a[j]==97+i)

{

count++;

}

}

c[i]=count;

}

for(i=0;i<26;i++)

{

j=97+i;

if(c[i]!=0) { printf("%c of %d times\n",j,c[i]);

}

}

getch();

}

1. Opening the file and counting and printing the number of times the letters are presents

#include<stdio.h>

#include<string.h>

#define filename "somefile.txt"

int main()

{

FILE \*fp;

int count[26] = {0}, i, c;

char ch;

char alpha[27] = "abcdefghijklmnopqrstuwxyz";

fp = fopen(filename,"r");

if(fp == NULL)

printf("file not found\n");

while( (ch = fgetc(fp)) != EOF) {

c = 0;

while(alpha[c] != '\0') {

if(alpha[c] == ch) {

count[c]++;

}

c++;

}

}

for(i = 0; i<26;i++) {

printf("character %c occured %d number of times\n",alpha[i], count[i]);

}

return 0;

}

18. Given a string and a sub string find out the number of ways the substring is present in the main string in different forms. Eg. Main string is “abckdaghcbajhbac” and the sub string “abc” you need to find out thenumber of times the sub string “abc” has occurred in different forms like “acb”, “bac”, “cab”,.. (6 possible ways of occurrence). For this case the output must be 3.

1. Inverse of the given string without using string function and another array.

#include<stdio.h>

#include<string.h>

int main() {

   char str[100], temp;

   int i, j = 0;

   printf("\nEnter the string :");

   gets(str);

   i = 0;

   j = strlen(str) - 1;

   while (i < j) {

      temp = str[i];

      str[i] = str[j];

      str[j] = temp;

      i++;

      j--;

   }

   printf("\nReverse string is :%s", str);

   return (0);

}

1. Find out second largest number in an unsorted array.

#include<stdio.h>

int main(){

  int a[50],size,i,j=0,big,secondbig;

  printf("Enter the size of the array: ");

  scanf("%d",&size);

  printf("Enter %d elements in to the array: ", size);

  for(i=0;i<size;i++)

      scanf("%d",&a[i]);

  big=a[0];

  for(i=1;i<size;i++){

      if(big<a[i]){

           big=a[i];

           j = i;

      }

  }

  secondbig=a[size-j-1];

  for(i=1;i<size;i++){

      if(secondbig <a[i] && j != i)

          secondbig =a[i];

  }

  printf("Second biggest: %d", secondbig);

  return 0;

}

**Sample output:**

Enter the size of the array: 5

Enter 5 elements in to the array: 5 3 2 1 0

Second biggest: 3

1. Find number of bits set (i.e 1) in a given integer. e.g. (input: 7 output: 3).

*/\**

*\* C Program to Count the Number of Bits set to One using*

*\* Bitwise Operations*

*\*/*

1. #include <stdio.h>
3. int main()
4. {
5. unsigned int number;
6. int count = 0;
8. printf("Enter the unsigned integer:**\n**");
9. scanf("%d", &number);
10. while (number != 0)
11. {
12. if ((number & 1) == 1)
13. count++;
14. number = number >> 1;
15. }
16. printf("number of one's are :**\n**%d**\n**", count);
17. return 0;
18. }

**1. Simple Method** Loop through all bits in an integer, check if a bit is set and if it is then increment the set bit count. See below program.

|  |
| --- |
| /\* Function to get no of set bits in binary     representation of passed binary no. \*/  int countSetBits(unsigned int n)  {    unsigned int count = 0;    while(n)    {      count += n & 1;      n >>= 1;    }    return count;  }    /\* Program to test function countSetBits \*/  int main()  {      int i = 9;      printf("%d", countSetBits(i));      getchar();      return 0;  } |

Run on IDE

**Time Complexity:** (-)(logn) (Theta of logn)

**2. Brian Kernighan’s Algorithm:**  
Subtraction of 1 from a number toggles all the bits (from right to left) till the rightmost set bit(including the righmost set bit). So if we subtract a number by 1 and do bitwise & with itself (n & (n-1)), we unset the righmost set bit. If we do n & (n-1) in a loop and count the no of times loop executes we get the set bit count.  
Beauty of the this solution is number of times it loops is equal to the number of set bits in a given integer.

1 Initialize count: = 0

2 **If** integer n is not zero

(a) Do bitwise & with (n-1) and assign the value back to n

n: = n&(n-1)

(b) Increment count by 1

(c) go to step 2

3 **Else** return count

**Implementation of Brian Kernighan’s Algorithm:**

|  |
| --- |
| #include<stdio.h>    /\* Function to get no of set bits in binary     representation of passed binary no. \*/  int countSetBits(int n)  {      unsigned int count = 0;      while (n)      {        n &= (n-1) ;        count++;      }      return count;  }    /\* Program to test function countSetBits \*/  int main()  {      int i = 9;      printf("%d", countSetBits(i));      getchar();      return 0;  } |

Run on IDE

**Example for Brian Kernighan’s Algorithm:**

n = 9 (1001)

count = 0

Since 9 > 0, subtract by 1 and do bitwise & with (9-1)

n = 9&8 (1001 & 1000)

n = 8

count = 1

Since 8 > 0, subtract by 1 and do bitwise & with (8-1)

n = 8&7 (1000 & 0111)

n = 0

count = 2

Since n = 0, return count which is 2 now.

1. Latin square matrix nxn

Input: 6

Output: 1 2 3 4 5 6

             2 3 4 5 6 1

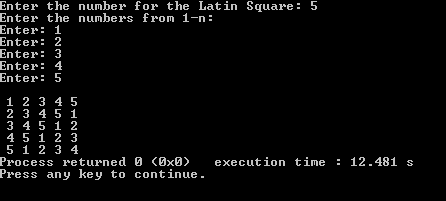
             3 4 5 6 1 2

             4 5 6 1 2 3

             5 6 1 2 3 4

             6 1 2 3 4 5

|  |
| --- |
| #include <stdio.h>  #include <math.h>    int main()  {      int i,j,k=1,n;      int range[n];      printf("Enter the number for the Latin Square: ");      scanf("%d",&n);      printf("Enter the numbers from 1-n: \n");      for(i=0;i<n;++i){          printf("Enter: ");          scanf("%d", &range[n]);          }      for (i=1;i<=n;i++){          printf("\n");          for (j=1;j<=n;j++)          {          printf(" %d",k);      if (k==n)      k=1;      else          k++;      }          k++;      }  return(0);  } |

1. This is what it outputs  
   
2. 10-08-2013
3. to find the permutation of any word eg:"abc" had to be coded

Below are the permutations of string ABC.  
ABC, ACB, BAC, BCA, CAB, CBA

Here is a solution using backtracking.

[](http://d2o58evtke57tz.cloudfront.net/wp-content/uploads/NewPermutation.gif)

# include<stdio.h>

# include<conio.h>

#include<string.h>

void swap (char \*x, char \*y)

{

    char temp;

    temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void permute(char \*a, int i, int n)

{

    int j;

    if (i == n)

    printf("%s\n", a);

    else

    {

        for (j = i; j <= n; j++)

        {

            swap((a+i), (a+j));

            permute(a, i+1, n);

            swap((a+i), (a+j));

        }

    }

}

int main()

{

    char str[80] ;

    int len=0,i;

    puts("Enter a string:\n\n");

    gets(str);

    for (i=0; str[i] != '\0'; i++)

    {

        len++;

    }

    permute(str, 0, len);

    getchar();

    return 0;

}

1. Reverse Each word in a String

Given a sentence and a pattern, find the number of occurences of the pattern in the sentence

1. display the elements from two arrays which do not belong to both the arrays

Method 1 (Naive)

Union:

1) Initialize union U as empty.

2) Copy all elements of first array to U.

3) Do following for every element x of second array:

…..a) If x is not present in first array, then copy x to U.

4) Return U.

Intersection:

1) Initialize intersection I as empty.

2) Do following for every element x of first array

…..a) If x is present in second array, then copy x to I.

4) Return I.

Time complexity of this method is O(mn) for both operations. Here m and n are number of elements in arr1[] and arr2[] respectively.

1. to display a pascal triangle pattern

void main()  
{  
int i,j,n,c,k,space;  
clrscr();  
printf("Enter the limit ");  
scanf("%d",&n);  
printf("\n\n");  
space=n;  
for(i=0;i<=n;i++)  
{  
c=1;  
for(k=space;k>=0;k--)  
printf(" ");  
  
space--;  
for(j=0;j<=i;j++)  
{  
printf("%d ",c);  
c=(c\*(i-j)/(j+1));  
}  
printf("\n");  
}  
getch();  
}

45.

Given a 2D

Matrix of chars and a string, find whether that string can be traced in

that matrix. u can move up, down, left or right between the chars in the matrix.

Example:

Matrix:

a o o f p d

f w l x r y

a k l o w a

String: follow

the program must print “true” bcoz

a o o f p d

f w l x r y

a k l o w a

#include<stdio.h>   
  
void main()   
{   
char matrix[10][10],find[10];   
int k=0,i1=0,j1=0,size=0,i,j,len,temp,chk=1 ;   
  
printf("Enter the size for char matrix for sizeXsize\n");   
scanf("%d",&size);   
  
printf("Enter the elements for char matrix \n");   
for(i=0;i<size;i++)   
{   
for(j=0;j<size;j++)   
{   
printf("\n Enter %dth row %dth col data\n",i+1,j+1);   
fflush(stdin);   
scanf("%c",&matrix[i][j]);   
}   
}   
printf("\n");   
  
for(i=0;i<size;i++)   
{   
for(j=0;j<size;j++)   
{   
printf("%c ",matrix[i][j]);   
}   
printf("\n");   
}   
  
printf("Word to be found out : \n");   
scanf("%s",find);   
len=strlen(find);   
temp=len;   
for(i=0;i<size && chk;i++)   
{   
for(j=0;j<size && chk;j++)   
{   
k=0;   
if(matrix[i][j]==find[k])   
{   
i1=i;   
j1=j;   
printf("\nFOUND\n");   
while(temp)   
{   
if(matrix[i1][j1]==find[k] &&   
i1<size &&   
j1 < size &&   
k <=len )   
{   
i1++;   
j1++;   
k++;   
}   
temp--;   
}   
  
if(i==(i1-len) && i1 < size && j1 < size && k <len)   
{   
printf("\nWORD exists diagonally\n");   
chk=0;   
break;   
}   
  
i1=i;   
j1=j;   
temp=len;   
k=0;   
while(temp)   
{   
if(matrix[i1][j1]==find[k] &&   
i1>=0 &&   
j1 >=0 &&   
k <=len )   
{   
i1--;   
j1--;   
k++;   
}   
temp--;   
}   
if(i==(i1+len))   
{   
printf("\nWORD exists diagonally upwards \n");   
chk=0;   
break;   
}   
  
  
i1=i;   
j1=j;   
k=0;   
temp=len;   
while(temp)   
{   
if(matrix[i1][j1]==find[k] &&   
i1<size &&   
j1 >=0 &&   
k <=len )   
{   
i1++;   
j1--;   
k++;   
}   
temp--;   
}   
  
if(i==(i1-len) && j==(j1+len))   
{   
printf("\nWORD exists across diagonally \n");   
chk=0;   
break;   
}   
  
k=0;   
i1=i;   
j1=j;   
temp=len;   
while(temp)   
{   
if(matrix[i1][j1]==find[k] && i1<size && j1 < size && k <=len)   
{   
i1++;   
k++;   
}   
temp--;   
}   
if(i==(i1-len))   
{   
printf("\nWORD exists vertically\n");   
chk=0;   
break;   
}   
  
k=0;   
i1=i;   
j1=j;   
temp=len;   
while(temp)   
{   
if(matrix[i1][j1]==find[k] && i1<size && j1 < size && k <=len && i1>=0 && j1 >=0 )   
{   
i1--;   
k++;   
}   
temp--;   
}   
if(i==(i1+len))   
{   
printf("\nWORD exists vertically upwards \n");   
chk=0;   
break;   
}   
  
k=0;   
i1=i;   
j1=j;   
temp=len;   
while(temp)   
{   
if(matrix[i1][j1]==find[k] && i1<size && j1 < size && k <=len)   
{   
j1++;   
k++;   
}   
temp--;   
}   
if(j==(j1-len))   
{   
printf("\nWORD exists horizentally\n");   
chk=0;   
break;   
}   
  
k=0;   
i1=i;   
j1=j;   
temp=len;   
while(temp)   
{   
if(matrix[i1][j1]==find[k] && i1<size && j1 < size && k <=len && i1>=0 && j1 >=0 )   
{   
j1--;   
k++;   
}   
temp--;   
}   
if(j==(j1+len))   
{   
printf("\nWORD exists horizentally upwards \n");   
chk=0;   
break;   
}   
}   
}   
}   
getch();   
}

46. . Two rectangles with cross dimensions are given. Write the condition to find whether both the rectangles are intersecting or not

First of all, let's define the rectangles and points in the following structures:

struct Point

{

int x;

int y;

};

struct Rectangle

{

Point topLeft; // Denotes the top-left point of the rectangle

Point bottomRight; // Denotes the bottom-right point of the rectangle

};

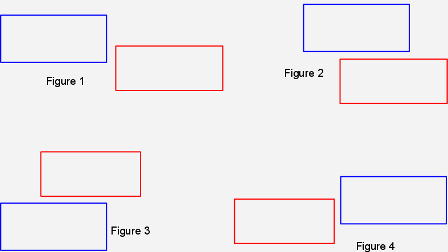
Given two rectangles R1 and R2 . It is easy to visualize that the given two rectangles can not be intersect if one of the following conditions is true.

Condition 1: When left edge of R1 is on the right of R2's right edge. ( That is , R1 is completely on the right of R2).

Condition 2: When right edge of R1 is on the left of R2's left edge. ( That is , R1 is completely on the left of R2).

Condition 3: When top edge of R1 is on bottom of R2's bottom edge ( That is , R1 is completely under R2).

Condition 4: When bottom edge of R1 is on top of R2's top edge ( That is , R1 is completely over R2).



Here is the following code based on the above conditions:

void RectIntersect( Rectangle R1 , Rectangle Rt2 )

{

if ( ( R1.topLeft.x > R2.bottomRight.x )|| ( R1.bottomRight.x < R2.topLeft.x ) || ( R1.topLeft.y > R2.bottomRight.y ) ||

( R1.bottomRight.y < R2.topLeft.y ) )

{

cout<<" Non - Intersecting";

}

else

{

cout<<" Intersecting";

}

}

We can also apply **De Morgan's law** :

**! ( Condition1 || Condition2 || Condition3 || Condition4 )** = **( ! Condition1 ) && ( ! Condition2 ) && ( ! Condition3 ) && ( ! Condition4 )**

Here is the following code based on the above conditions:

void RectIntersect( Rectangle R1 , Rectangle R2 )

{

if ( ( R1.topLeft.x < R2.bottomRight.x ) && ( R1.bottomRight.x > R2.topLeft.x ) &&

( R1.topLeft.y < R2.bottomRight.y ) &&

( R1.bottomRight.y > R2.topLeft.y ) )

{

cout<<"Intersecting"

}

else

{

cout<<" Non - Intersecting";

}

}

1. Given a number in roman representation. Convert it into normal representation.

Given roman symbols with their values are

I-1

V-5

X-10

L-50

C-100

Input: CVI-106

XIV-14

Step 1: Start

Step 2: read the roman numerical as string

Step 3: find length of roman numerical

Step 4: for each charcter in the string

i) if(char = I) then decimal = 1

ii) if(char = V) then decimal = 5

iii) if(char = X) then decimal = 10

iv) if(char = L) then decimal = 50

v) if(char = C) then decimal = 100

vi) if(char = D) then decimal = 500

vii) if(char = M) then decimal = 1000

viii) otherwise invalid character

Step 5: repeat step 4 until the length of the string

Step 6: k = char[length - 1]

Step 7: for each character of decimal string

i) if(decimal[i] > dec[i - 1]) then k = k - decimal[i - 1]

ii) else if(decimal[i] = decimal[i - 1 or decimal[i] < decimal[i - 1) then k = k + decimall[i - 1]

Step 8: repate step 7 until the length of decimal string

Step 9: print decimal value

Step 10: Stop

Program:

#include <stdio.h>

#include <conio.h>

#include <string.h>

#include <stdlib.h>

void main()

{

char rom[30];

int a[30], l, i, k, dec;

clrscr();

printf("Enter the roman number\n");

scanf("%s", &rom);

l =strlen(rom);

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

{

switch (rom[i])

{

case 'I': a[i] = 1;

break;

case 'V': a[i] = 5;

break;

case 'X': a[i] = 10;

break;

case 'L': a[i] = 50;

break;

case 'C': a[i] = 100;

break;

case 'D': dec = dec + 500;

break;

case 'M': a[i] = 1000;

break;

default : printf("Invalid choice");

break;

}

}

k = a[l - 1];

for(i = l - 1; i > 0; i--)

{

if(a[i] > a[i - 1])

{

k = k - a[i - 1];

}

if(a[i] <= a[i - 1])

{

k = k + a[i - 1];

}

}

printf("decimal equivalent is %d", k);

getch();

}

Input & Output:

Enter the roman number

XIV

Decimal equivalent is 14

1. Given a sentence and every first letter of all words should be in capital letter.

Input: zoHO coRporation 1st recruitment

Output: Zoho Corporation 1st Recruitment

#include <stdio.h>

#include <string.h>

capital(char s[])

{

int i;

for(i=0; i<strlen(s); i++)

{

if (i==0||s[i-1]==' '&&s[i]>='a'&&s[i]<='z')

s[i]=toupper(s[i]);

}

puts(s);

}

main()

{

char s[100];

printf("Enter a line: ");

gets(s);

capital(s);

}

1. Check whether a string is balanced or not. Eg: ip={(}) op=not balanced ip={(())} op=balanced

Algorithm:  
1) Declare a character stack S.  
2) Now traverse the expression string exp.  
    a) If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack.  
    b) If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.  
3) After complete traversal, if there is some starting bracket left in stack then “not balanced”

Implementation:

|  |
| --- |
| #include<stdio.h>  #include<stdlib.h>  #define bool int    /\* structure of a stack node \*/  struct sNode  {     char data;     struct sNode \*next;  };    /\* Function to push an item to stack\*/  void push(struct sNode\*\* top\_ref, int new\_data);    /\* Function to pop an item from stack\*/  int pop(struct sNode\*\* top\_ref);    /\* Returns 1 if character1 and character2 are matching left     and right Parenthesis \*/  bool isMatchingPair(char character1, char character2)  {     if (character1 == '(' && character2 == ')')       return 1;     else if (character1 == '{' && character2 == '}')       return 1;     else if (character1 == '[' && character2 == ']')       return 1;     else       return 0;  }    /\*Return 1 if expression has balanced Parenthesis \*/  bool areParenthesisBalanced(char exp[])  {     int i = 0;       /\* Declare an empty character stack \*/     struct sNode \*stack = NULL;       /\* Traverse the given expression to check matching parenthesis \*/     while (exp[i])     {        /\*If the exp[i] is a starting parenthesis then push it\*/        if (exp[i] == '{' || exp[i] == '(' || exp[i] == '[')          push(&stack, exp[i]);          /\* If exp[i] is a ending parenthesis then pop from stack and            check if the popped parenthesis is a matching pair\*/        if (exp[i] == '}' || exp[i] == ')' || exp[i] == ']')        {              /\*If we see an ending parenthesis without a pair then return false\*/           if (stack == NULL)             return 0;             /\* Pop the top element from stack, if it is not a pair              parenthesis of character then there is a mismatch.              This happens for expressions like {(}) \*/           else if ( !isMatchingPair(pop(&stack), exp[i]) )             return 0;        }        i++;     }       /\* If there is something left in expression then there is a starting        parenthesis without a closing parenthesis \*/     if (stack == NULL)       return 1; /\*balanced\*/     else       return 0;  /\*not balanced\*/  }    /\* UTILITY FUNCTIONS \*/  /\*driver program to test above functions\*/  int main()  {    char exp[100] = "{()}[]";    if (areParenthesisBalanced(exp))      printf("\n Balanced ");    else      printf("\n Not Balanced ");    return 0;  }    /\* Function to push an item to stack\*/  void push(struct sNode\*\* top\_ref, int new\_data)  {    /\* allocate node \*/    struct sNode\* new\_node =              (struct sNode\*) malloc(sizeof(struct sNode));      if (new\_node == NULL)    {       printf("Stack overflow \n");       getchar();       exit(0);    }      /\* put in the data  \*/    new\_node->data  = new\_data;      /\* link the old list off the new node \*/    new\_node->next = (\*top\_ref);      /\* move the head to point to the new node \*/    (\*top\_ref)    = new\_node;  }    /\* Function to pop an item from stack\*/  int pop(struct sNode\*\* top\_ref)  {    char res;    struct sNode \*top;      /\*If stack is empty then error \*/    if (\*top\_ref == NULL)    {       printf("Stack overflow \n");       getchar();       exit(0);    }    else    {       top = \*top\_ref;       res = top->data;       \*top\_ref = top->next;       free(top);       return res;    }  } |