**Number of Digits**

**Send Feedback**

Given the code to find out and return the number of digits present in a number recursively. But it contains few bugs, that you need to rectify such that all the test cases should pass.

**Input Format :**

Integer n

**Output Format :**

Count of digits

**Constraints :**

1 <= n <= 10^6

**Sample Input 1 :**

156

**Sample Output 1 :**

3

**Sample Input 2 :**

7

**Sample Output 2 :**

1

Sol

int count(int n){

if(n == 0)

{

return 0;

}

int smallAns = count(n / 10);

return smallAns + 1;

}

Alternate:

If 0 <= n <= 10^6

int func(int n){

if(n==0)

return 1;

int last=n%10;

int rem=n/10;

if(n/10==0)

return func(rem);

else

return func(rem)+1;

}

Toh

Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move all disks from source rod to destination rod using third rod (say auxiliary). The rules are :

1) Only one disk can be moved at a time.

2) A disk can be moved only if it is on the top of a rod.

3) No disk can be placed on the top of a smaller disk.

Print the steps required to move n disks from source rod to destination rod.

Source Rod is named as 'a', auxiliary rod as 'b' and destination rod as 'c'.

**Input Format :**

Integer n

**Output Format :**

Steps in different lines (in one line print source and destination rod name separated by space)

**Constraints :**

0 <= n <= 20

**Sample Input 1 :**

2

**Sample Output 1 :**

a b

a c

b c

**Sample Input 2 :**

3

**Sample Output 2 :**

a c

a b

c b

a c

b a

b c

a c

Sol

void towerOfHanoi(int n, char src, char aux, char dest) {

// Write your code here

if(n==0)

return;

towerOfHanoi(n-1,src,dest,aux);

cout<<src<<" "<<dest<<endl;

towerOfHanoi(n-1,aux,src,dest);

}

Alternative:

int toh(int n,char src,char aux,char dest){

if(n==0)

return 0;

int c=toh(n-1,src,dest,aux);

cout<<src<<" "<<dest<<endl;

int r=toh(n-1,aux,src,dest);

return c+r+1;

}

int main()

{

cout<<toh(3,'a','b','c');

return 0;

}

**First Index of Number**

**Send Feedback**

Given an array of length N and an integer x, you need to find and return the first index of integer x present in the array. Return -1 if it is not present in the array.

First index means, the index of first occurrence of x in the input array.

Do this recursively. Indexing in the array starts from 0.

**Input Format :**

Line 1 : An Integer N i.e. size of array

Line 2 : N integers which are elements of the array, separated by spaces

Line 3 : Integer x

**Output Format :**

first index or -1

**Constraints :**

1 <= N <= 10^3

**Sample Input :**

4

9 8 10 8

8

**Sample Output :**

1

Sol

int firstIndex(int input[], int size, int x) {

/\* Don't write main().

Don't read input, it is passed as function argument.

Return output and don't print it.

Taking input and printing output is handled automatically.

\*/

if(size==1)

return (input[0]==x?0:-1);

if(input[0]==x)

return 0;

int ans=firstIndex(input+1,size-1,x);

if(ans==-1)

return -1;

return 1+ans;

}

**Last Index of Number**

**Send Feedback**

Given an array of length N and an integer x, you need to find and return the last index of integer x present in the array. Return -1 if it is not present in the array.

Last index means - if x is present multiple times in the array, return the index at which x comes last in the array.

You should start traversing your array from 0, not from (N - 1).

Do this recursively. Indexing in the array starts from 0.

**Input Format :**

Line 1 : An Integer N i.e. size of array

Line 2 : N integers which are elements of the array, separated by spaces

Line 3 : Integer x

**Output Format :**

last index or -1

**Constraints :**

1 <= N <= 10^3

**Sample Input :**

4

9 8 10 8

8

**Sample Output :**

3

Sol

int lastIndex(int input[], int size, int x) {

/\* Don't write main().

Don't read input, it is passed as function argument.

Return output and don't print it.

Taking input and printing output is handled automatically.

\*/

if(size==0)

return -1;

if(input[size-1]==x)

return size-1;

return lastIndex(input, size-1,x);

}

**All Indices of Number**

**Send Feedback**

Given an array of length N and an integer x, you need to find all the indexes where x is present in the input array. Save all the indexes in an array (in increasing order).

Do this recursively. Indexing in the array starts from 0.

**Input Format :**

Line 1 : An Integer N i.e. size of array

Line 2 : N integers which are elements of the array, separated by spaces

Line 3 : Integer x

**Output Format :**

indexes where x is present in the array (separated by space)

**Constraints :**

1 <= N <= 10^3

**Sample Input :**

5

9 8 10 8 8

8

**Sample Output :**

1 3 4

Sol

/\*

Given an array of length N and an integer x, you need to find all the indexes where x is present in the input array. Save all the indexes in an array (in increasing order).

Do this recursively. Indexing in the array starts from 0.

\*/

#include<iostream>

using namespace std;

//forward cutting

int allIndexes2(int input[], int size, int x, int output[])

{

if(input==NULL)

return -1;

if(size==1)

{

int top=0;

if(input[0]==x)

{

output[0]=0;

top++;

}

return top;

}

int top=allIndexes2(input+1,size-1,x,output);

// printf("%d",top);

for(int i=0;i<top;i++)

{

output[i]++;

}

if(input[0]==x)

{

for(int i=top;i>0;i--)

{

output[i]=output[i-1];

}

output[0]=0;

top++;

}

return top;

}

//backward cutting

int allIndexes(int input[], int size, int x, int output[])

{

if(input==NULL)

return -1;

if(size==1)//same function as above

{

int top=0;

if(input[size-1]==x)

{output[top++]=size-1;return top;}

return top;

}

int top=allIndexes(input,size-1,x,output);

//printf("%d",top);

if(input[size-1]==x)

{output[top++]=size-1;return top;}

return top;

}

/\*

int main(){

int n;

cin >> n;

int \*input = new int[n];

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

cin >> input[i];

}

int x;

cin >> x;

int \*output = new int[n];

int size = allIndexes(input, n, x, output);

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

cout << output[i] << " ";

}

delete [] input;

delete [] output;

}

\*/

**Check Palindrome (recursive)**

**Send Feedback**

Check whether a given String S is a palindrome using recursion. Return true or false.

**Input Format :**

String S

**Output Format :**

'true' or 'false'

**Constraints :**

0 <= |S| <= 1000

where |S| represents length of string S.

**Sample Input 1 :**

racecar

**Sample Output 1:**

true

**Sample Input 2 :**

ninja

**Sample Output 2:**

false

Sol

bool checkPalindrome(char input[])

{

int length=0;

for(int i=0;input[i]!='\0';i++)

length++;

if(length==0)//handles both length of string odd and even

return true;

if(input[0]==input[length-1])

{

input[length-1]='\0';

return checkPalindrome(input+1);

}

return false;

}

Alternative:

void func(string str){

if(str.length()==0)

{cout<<"True";

return;}

if(str[0]==str[str.length()-1])

func(str.substr(1,str.length()-2));

else

{

cout<<"False";return;

}

}

**Geometric Sum**

**Send Feedback**

Given k, find the geometric sum i.e.

1 + 1/2 + 1/4 + 1/8 + ... + 1/(2^k)

using recursion.

**Input format :**

Integer k

**Output format :**

Geometric sum

**Constraints :**

0 <= k <= 1000

**Sample Input 1 :**

3

**Sample Output 1 :**

1.875

**Sample Input 2 :**

4

**Sample Output 2 :**

1.93750

Sol

double geometricSum(int k) {

// Write your code here

if(k==0)

return 1;

return 1.0/pow(2,k)+geometricSum(k-1);

}

**Sum of Array**

**Send Feedback**

Given an array of length N, you need to find and return the sum of all elements of the array.

Do this recursively.

**Input Format :**

Line 1 : An Integer N i.e. size of array

Line 2 : N integers which are elements of the array, separated by spaces

**Output Format :**

Sum

**Constraints :**

1 <= N <= 10^3

**Sample Input 1 :**

3

9 8 9

**Sample Output 1 :**

26

**Sample Input 2 :**

3

4 2 1

**Sample Output 2 :**

7

Sol

int sum(int input[], int n) {

/\* Don't write main().

Don't read input, it is passed as function argument.

Return output and don't print it.

Taking input and printing output is handled automatically.

\*/

if(input==NULL)

return -1;

if(n==1)

return input[0];

return input[0]+sum(input+1,n-1);

}

**Check Number**

**Send Feedback**

Given an array of length N and an integer x, you need to find if x is present in the array or not. Return true or false.

Do this recursively.

**Input Format :**

Line 1 : An Integer N i.e. size of array

Line 2 : N integers which are elements of the array, separated by spaces

Line 3 : Integer x

**Output Format :**

'true' or 'false'

**Constraints :**

1 <= N <= 10^3

**Sample Input 1 :**

3

9 8 10

8

**Sample Output 1 :**

true

**Sample Input 2 :**

3

9 8 10

2

**Sample Output 2 :**

false

Sol

bool checkNumber(int input[], int size, int x) {

/\* Don't write main().

Don't read input, it is passed as function argument.

Return output and don't print it.

Taking input and printing output is handled automatically.

\*/

if(input==NULL)

return false;

if(size==1)

return input[0]==x;

if(input[0]==x)

return true;

return checkNumber(input+1,size-1,x);

}

**Sum of digits (recursive)**

**Send Feedback**

Write a recursive function that returns the sum of the digits of a given integer.

**Input format :**

Integer N

**Output format :**

Sum of digits of N

**Constraints :**

0 <= N <= 10^9

**Sample Input 1 :**

12345

**Sample Output 1 :**

15

**Sample Input 2 :**

9

**Sample Output 2 :**

9

Sol

int sumOfDigits(int n) {

// Write your code here

if(n==0)

return 0;

int r=n%10;

return r+sumOfDigits(n/10);

}

**String to Integer**

**Send Feedback**

Write a recursive function to convert a given string into the number it represents. That is input will be a numeric string that contains only numbers, you need to convert the string into corresponding integer and return the answer.

**Input format :**

Numeric string S (string, Eg. "1234")

**Output format :**

Corresponding integer N (int, Eg. 1234)

**Constraints :**

0 <= |S| <= 9

where |S| represents length of string S.

**Sample Input 1 :**

1231

**Sample Output 1 :**

1231

**Sample Input 2 :**

12567

**Sample Output 2 :**

12567

Sol

int pow(char st[])

{

int prod=1;

for(int i=0;st[i]!='\0';i++)

prod=prod\*10;

return prod/10;

}

int stringToNumber(char input[]) {

if(input[0]=='\0')

return 0;

int msd=((int)input[0]-48)\*pow(input);

int ans=msd+stringToNumber(input+1);

if(input[0]=='-')

{

ans-=((int)input[0]-48)\*pow(input);

ans=-ans;

}

return ans;

}

Alternative:

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

int func(string str){

if(str.length()==0)

return -1;

if(str.length()==1)

return str[0]-'0';

int first=str[0]-'0';

int num=func(str.substr(1,str.length()-1));

return first\*(ceil(pow(10,str.length()-1)))+num;

}

int main()

{

string str="12353534";

cout<<func(str);

return 0;

}

**Pair star**

**Send Feedback**

Given a string S, compute recursively a new string where identical chars that are adjacent in the original string are separated from each other by a "\*".

**Input format :**

String S

**Output format :**

Modified string

**Constraints :**

0 <= |S| <= 1000

where |S| represents length of string S.

**Sample Input 1 :**

hello

**Sample Output 1:**

hel\*lo

**Sample Input 2 :**

aaaa

**Sample Output 2 :**

a\*a\*a\*a

Sol

// Change in the given string itself. So no need to return or print the changed string.

int length(char s[])

{

int l=0;

for(int i=0;s[i]!='\0';i++)

l++;

//printf("%d\n",l);

return l;

}

void pairStar(char input[]) {

// Write your code here

int skip=1;

if(input[0]=='\0')

return;

if(input[0]==input[1])

{

int l=length(input);

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

{

input[i+1]=input[i];

}

input[1]='\*';

skip=2;

}

pairStar(input+skip);

}

**Replace pi (recursive)**

**Send Feedback**

Given a string, compute recursively a new string where all appearances of "pi" have been replaced by "3.14".

**Sample Input 1 :**

xpix

**Sample Output :**

x3.14x

**Sample Input 2 :**

pipi

**Sample Output :**

3.143.14

**Sample Input 3 :**

pip

**Sample Output :**

3.14p

Sol

// Change in the given string itself. So no need to return or print anything

int length(char s[])

{

int len=0;

for(int i=0;s[i]!='\0';i++)

len++;

return len;

}

void replacePi(char input[])

{

// Write your code here

if(input[0]=='\0')

return;

if(input[0]=='p' && input[1]=='i')

{

int len=length(input);

for(int i=len;i>=0;i--)//l+1 times

{

input[i+2]=input[i];

}

input[0]='3';

input[1]='.';

input[2]='1';

input[3]='4';

replacePi(input+3);

}

replacePi(input+1);

}

**Does s contain t ?**

**Send Feedback**

#### Given two string s and t, write a function to check if s contains all characters of t (in the same order as they are in string t).

#### Return true or false.

#### Do it recursively.

#### E.g. : s = “abchjsgsuohhdhyrikkknddg” contains all characters of t=”coding” in the same order. So function will return true.

##### Input Format :

Line 1 : String s

Line 2 : String t

##### Output Format :

true or false

##### Sample Input 1 :

abchjsgsuohhdhyrikkknddg

coding

##### Sample Output 1 :

true

##### Sample Input 2 :

abcde

aeb

##### Sample Output 2 :

false

Sol:

bool checksequenece(char large[] , char\*small) {

if(large[0]=='\0'){

if(small[0]=='\0')

return true;

return false;

}

if(large[0]==small[0]){

return checksequenece(large+1,small+1);

}

return checksequenece(large+1,small);

}