**Replace Character Recursively**

**Send Feedback**

Given an input string S and two characters c1 and c2, you need to replace every occurrence of character c1 with character c2 in the given string.

Do this recursively.

**Input Format :**

Line 1 : Input String S

Line 2 : Character c1 and c2 (separated by space)

**Output Format :**

Updated string

**Constraints :**

1 <= Length of String S <= 10^6

**Sample Input :**

abacd

a x

**Sample Output :**

xbxcd

Sol

void replaceCharacter(char input[], char ch1, char ch2) {

/\* Don't write main().

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

\* No need to print or return the output.

\* Change in the given input string itself.

\* Taking input and printing output is handled automatically.

\*/

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

return;

if(input[0]==ch1)

input[0]=ch2;

replaceCharacter(input+1,ch1,ch2);

}

Solution2: with strings

#include <iostream>

#include <string>

using namespace std;

string func(string str,char ch1,char ch2){

if(str.length()==0)

return "";

string temp=func(str.substr(1),ch1,ch2);

// cout<<"temp:"<<temp<<",str[0]:"<<str[0]<<endl;

if(str[0]==ch1)

return ch2+temp;

return str[0]+temp;

}

int main()

{

string str="abcad";

cout<<func(str,'a','x');

}

**Remove Duplicates Recursively**

**Send Feedback**

Given a string S, remove consecutive duplicates from it recursively.

**Input Format :**

String S

**Output Format :**

Output string

**Constraints :**

1 <= |S| <= 10^3

where |S| represents the length of string

**Sample Input 1 :**

aabccba

**Sample Output 1 :**

abcba

**Sample Input 2 :**

xxxyyyzwwzzz

**Sample Output 2 :**

xyzwz

Sol

void removeConsecutiveDuplicates(char \*input) {

/\* Don't write main().

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

\* Change in the given string itself.

\* No need to return or print anything

\* Taking input and printing output is handled automatically.

\*/

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

return;

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

{

int i=2;

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

input[i-1]=input[i];

input[i-1]=input[i];//copying null character

removeConsecutiveDuplicates(input);

}

else

removeConsecutiveDuplicates(input+1);

}

Solution 2:with strings

#include <iostream>

#include <string>

using namespace std;

string func(string str){

if(str.length()==0)

return "";

if(str[0]==str[1]){

str=str.substr(1);

return func(str);

}

string temp=func(str.substr(1));

return str[0]+temp;

}

int main()

{

string str="xxxyyyzwwzzz";

cout<<func(str);

}

**Merge Sort Code**

**Send Feedback**

Sort an array A using Merge Sort.

Change in the input array itself. So no need to return or print anything.

**Input format :**

Line 1 : Integer n i.e. Array size

Line 2 : Array elements (separated by space)

**Output format :**

Array elements in increasing order (separated by space)

**Constraints :**

1 <= n <= 10^3

**Sample Input 1 :**

6

2 6 8 5 4 3

**Sample Output 1 :**

2 3 4 5 6 8

**Sample Input 2 :**

5

2 1 5 2 3

**Sample Output 2 :**

1 2 2 3 5

Sol

void merge(int input[],int start, int end)

{

int mid=(start+end)/2;

int size=end-start+1;

int \*temp=new int[size];int top=-1;

int i=start, j=mid+1;

while(i<=mid && j<=end)

{

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

{

temp[++top]=input[i];

i++;

}

else

{

temp[++top]=input[j];

j++;

}

}

while(i<=mid)

{

temp[++top]=input[i];

i++;

}

while(j<=end)

{

temp[++top]=input[j];

j++;

}

int m=0;

for(int i=start;i<=end;i++)

{

input[i]=temp[m];

m++;

}

delete [] temp;

}

void mergeSort(int input[], int size)

{

if(size<=1)

return;

int st=0;

int end=size-1;

int mid=end/2;

mergeSort(input, mid+1);

mergeSort(input+mid+1,end-mid);

merge(input,st,end);

}

Solution 2: standard method

#include <iostream>

#include <string>

using namespace std;

void merge(int arr[],int start,int end){

int mid=(start+end)/2;

int i=start,j=mid+1;

int \*temparr=new int[end-start+1];

int m=0;

while(i<=mid && j<=end){

if(arr[i]<=arr[j]){

temparr[m]=arr[i];

i++;

}

else {

temparr[m]=arr[j];

j++;

}

m++;

}

while(i<=mid){

temparr[m]=arr[i];

i++;

m++;

}

while(j<=end){

temparr[m]=arr[j];

j++;

m++;

}

//copying

int r=0;

for(int y=start;y<=end;y++){

arr[y]=temparr[r];

r++;

}

delete []temparr;

}

void mergeSort(int arr[],int start, int end){

if(start<end)

{

int mid=(start+end)/2;

mergeSort(arr,start,mid);

mergeSort(arr,mid+1,end);

merge(arr,start,end);

}

}

void mergeSort(int arr[],int n){

mergeSort(arr,0,n-1);

}

int main()

{

int arr[6]={2,6,8,5,4,3};

int n=sizeof(arr)/sizeof(int);

mergeSort(arr,n);

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

cout<<arr[i]<<" ";

}

**Quick Sort Code**

**Send Feedback**

Sort an array A using Quick Sort.

Change in the input array itself. So no need to return or print anything.

**Input format :**

Line 1 : Integer n i.e. Array size

Line 2 : Array elements (separated by space)

**Output format :**

Array elements in increasing order (separated by space)

**Constraints :**

1 <= n <= 10^3

**Sample Input 1 :**

6

2 6 8 5 4 3

**Sample Output 1 :**

2 3 4 5 6 8

**Sample Input 2 :**

5

1 5 2 7 3

**Sample Output 2 :**

1 2 3 5 7

Sol

int partition(int input[],int end)

{

int x=input[0];

int index=0;

for(int i=1;i<=end;i++)

{

if(input[i]<x)

index++;

}

//swapping

int temp=input[index];

input[index]=x;

input[0]=temp;

//segregating

int i=0,j=end;

while(i!=index && j!=index)

{

if(input[i]<x)

i++;

else if(input[j]>=x)

j--;

else

{

temp=input[i];

input[i]=input[j];

input[j]=temp;

i++;

j--;

}

}

// return index;

}

void quickSort(int input[], int size) {

/\* Don't write main().

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

Change in the given array itself.

Taking input and printing output is handled automatically.

\*/

if(size<=1)

return;

int index=partition(input,size-1);

quickSort(input,index);

quickSort(input+index+1,size-1 - index);

}

Solution 2:

int partition(int arr[],int start,int end){

int count=0;

int pivot=arr[start];

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

if(arr[i]<pivot)

count++;

}

swap(arr[count],arr[start]);

int i=start;

int j=end;

while(i<count && j>count){

if(arr[i]<pivot)

i++;

else if(arr[j]>=pivot)

j--;

else {

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

i++;

j--;

}

}

return count;

}

void quickSort(int arr[],int start, int end){

if(start<end){

int size=end-start+1;

// cout<<start<<end<<endl;

int index=partition(arr,start,end);

quickSort(arr,start,index-1);

quickSort(arr,index+1,end);

}

}

void quickSort(int arr[],int size){

quickSort(arr,0,size-1);

}

int main()

{

int arr[6]={2, 6, 8, 5, 4, 3};

int n=sizeof(arr)/sizeof(int);

quickSort(arr,n);

// cout<<partition(arr,0,6);

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

cout<<arr[i]<<" ";

}

**Return subset of an array**

**Send Feedback**

Given an integer array (of length n), find and return all the subsets of input array.

**Subsets are of length varying from 0 to n, that contain elements of the array. But the order of elements should remain same as in the input array.**

**Note : The order of subsets are not important.**

Input format :

Line 1 : Size of array

Line 2 : Array elements (separated by space)

**Sample Input:**

3

15 20 12

**Sample Output:**

[] (this just represents an empty array, don't worry about the square brackets)

12

20

20 12

15

15 12

15 20

15 20 12

/\*\*\*

You need to save all the subsets in the given 2D output array. And return the number of subsets(i.e. number of rows filled in output) from the given function.

In ith row of output array, 1st column contains length of the ith subset. And from 1st column actual subset follows.

For eg. Input : {1, 2}, then output should contain

{{0}, // Length of this subset is 0

{1, 2}, // Length of this subset is 1

{1, 1}, // Length of this subset is also 1

{2, 1, 2}} // Length of this subset is 2

Don’t print the subsets, just save them in output.

\*\*\*/

int subset(int input[], int n, int output[][20]) {

// Write your code here

if(n==0){

output[0][0]=0;

return 1;

}

int len=subset(input+1,n-1,output);

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

for(int j=1;j<=output[i][0];j++){

output[len+i][j+1]=output[i][j];

}

output[len+i][0]=output[i][0]+1;

output[len+i][1]=input[0];

}

return 2\*len;

}

**Print Subsets of Array**

**Send Feedback**

Given an integer array (of length n), find and print all the subsets of input array.

**Subsets are of length varying from 0 to n, that contain elements of the array. But the order of elements should remain same as in the input array.**

**Note : The order of subsets are not important. Just print the subsets in different lines.**

**Input format :**

Line 1 : Integer n, Size of array

Line 2 : Array elements (separated by space)

**Constraints :**

1 <= n <= 15

**Sample Input:**

3

15 20 12

**Sample Output:**

[] (this just represents an empty array, don't worry about the square brackets)

12

20

20 12

15

15 12

15 20

15 20 12

Sol

void print(int arr[],int size)

{

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

cout<<arr[i]<<" ";

cout<<endl;

}

void psoa(int input[],int start, int size,int output[], int top)

{

if(start==size)

{

print(output,top);

return;

}

psoa(input,start+1,size,output,top);

output[top++]=input[start];

psoa(input,start+1,size,output,top);

}

void printSubsetsOfArray(int input[], int size) {

// Write your code here

int \*output=new int[size];

psoa(input,0,size,output,0);

delete [] output;

}

Solution 2:Easier method

#include <iostream>

#include <string>

using namespace std;

void func(int arr[],int size, int output[], int osize){

if(size<=0){

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

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

cout<<endl;

return;

}

func(arr+1,size-1, output, osize);

output[osize]=arr[0];

func(arr+1,size-1, output, osize+1);

}

void func(int arr[], int size){

int \*output=new int[size];

int osize=0;

func(arr,size,output,osize);

delete[]output;

}

int main()

{

int arr[3]={15, 20, 12};

int n=sizeof(arr)/sizeof(int);

func(arr,n);

}

**Return subsets sum to K**

**Send Feedback**

Given an array A of size n and an integer K, return all subsets of A which sum to K.

**Subsets are of length varying from 0 to n, that contain elements of the array. But the order of elements should remain same as in the input array.**

**Note : The order of subsets are not important.**

**Input format :**

Line 1 : Integer n, Size of input array

Line 2 : Array elements separated by space

Line 3 : K

**Constraints :**

1 <= n <= 20

**Sample Input :**

9

5 12 3 17 1 18 15 3 17

6

**Sample Output :**

3 3

5 1

Sol

/\*\*\*

You need to save all the subsets in the given 2D output array. And return the number of subsets(i.e. number of rows filled in output) from the given function.

In ith row of output array, 1st column contains length of the ith subset. And from 1st column actual subset follows.

For eg. Input : {1, 3, 4, 2} and K = 5, then output array should contain

{{2, 1, 4}, // Length of this subset is 2

{2, 3, 2}} // Length of this subset is 2

Don’t print the subsets, just save them in output.

\*\*\*/

int checkSum(int temp[],int top,int output[][50], int k, int s\_top)

{

int sum=0;

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

sum+=temp[i];

if(sum!=k)

return 1;

output[s\_top][0]=top;

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

output[s\_top][i+1]=temp[i];

return 0;

}

int subsetSumToK(int input[], int fixed, int n, int temp[], int top, int output[][50], int k) {

// Write your code here

static int s\_top;

if(fixed==n)

{

int r=checkSum(temp,top,output,k,s\_top);

if(r==0)

s\_top++;

return 0;

}

subsetSumToK(input,fixed+1,n,temp,top,output,k);

temp[top++]=input[fixed];

subsetSumToK(input,fixed+1,n,temp,top,output,k);

return s\_top;

}

int subsetSumToK(int input[], int n, int output[][50], int k) {

int \*temp=new int[n];

int t = subsetSumToK(input, 0, n, temp, 0, output, k);

delete []temp;

return t;

}

**Print Subset Sum to K**

**Send Feedback**

Given an array A and an integer K, print all subsets of A which sum to K.

**Subsets are of length varying from 0 to n, that contain elements of the array. But the order of elements should remain same as in the input array.**

**Note : The order of subsets are not important. Just print them in different lines.**

**Input format :**

Line 1 : Size of input array

Line 2 : Array elements separated by space

Line 3 : K

**Sample Input:**

9

5 12 3 17 1 18 15 3 17

6

**Sample Output:**

3 3

5 1

Sol

void printSubsetSumToK(int input[], int size, int k, int output[], int top) {

// Write your code here

if(size==0)

{

if(k==0){

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

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

cout<<endl;}

return;

}

printSubsetSumToK(input+1, size-1,k,output,top);

output[top]=input[0];

top++;

printSubsetSumToK(input+1,size-1,k-input[0],output,top);

}

void printSubsetSumToK(int input[], int size, int k)

{

int \*output=new int[size];

printSubsetSumToK(input, size, k, output, 0) ;

delete[]output;

}

**Return all codes - String**

**Send Feedback**

Assume that the value of a = 1, b = 2, c = 3, ... , z = 26. You are given a numeric string S. Write a program to return the list of all possible codes that can be generated from the given string.

**Note : The order of codes are not important. And input string does not contain 0s.**

**Input format :**

A numeric string

**Constraints :**

1 <= Length of String S <= 10

**Sample Input:**

1123

**Sample Output:**

aabc

kbc

alc

aaw

kw

Sol

#include <string.h>

using namespace std;

char ascii(int i){

if(i>0 && i<27)

return char(96+i);

return '\0';

}

int getCodes(string input, string output[10000]) {

/\*

You are given the input text and output string array. Find all possible codes and store in the output string array. Don’t print the codes.

Also, return the number of codes return to the output string. You do not need to print anything.

\*/

if(input.empty()){

output[0]="";

return 1;

}

string smalloutput1[10000],smalloutput2[10000];

int s1=0,s2=0;

char ch1=ascii(input[0]-48);

char ch2='\0';

s1=getCodes(input.substr(1), smalloutput1);

if(input[1]!='\0')

{

int num=(input[0]-48)\*10+(input[1]-48);

ch2=ascii(num);

// cout<<num<<ch2<<endl;

if(ch2!='\0')

s2=getCodes(input.substr(2), smalloutput2);

}

int k=0;

for(int i=0;i<s1;i++)

{

output[k]=ch1+smalloutput1[i];

k++;

}

for(int i=0;i<s2;i++)

{

output[k]=ch2+smalloutput2[i];

k++;

}

return k;

}

**Print all Codes - String**

**Send Feedback**

Assume that the value of a = 1, b = 2, c = 3, ... , z = 26. You are given a numeric string S. Write a program to print the list of all possible codes that can be generated from the given string.

**Note : The order of codes are not important. Just print them in different lines.**

**Input format :**

A numeric string S

**Output Format :**

All possible codes in different lines

**Constraints :**

1 <= Length of String S <= 10

**Sample Input:**

1123

**Sample Output:**

aabc

kbc

alc

aaw

kw

Sol

#include <string.h>

using namespace std;

char ascii(int i){

if(i>0 && i<27)

return char(96+i);

return '\0';

}

void printAllPossibleCodes(string input, string temp)

{

if(input.length()==0)

{

cout<<temp<<endl;

return;

}

char ch1=ascii(input[0]-48);

char ch2='\0';

printAllPossibleCodes(input.substr(1), temp+ch1);

if(input[1]!='\0')

{

int num=(input[0]-48)\*10+(input[1]-48);

ch2=ascii(num);

}

if(ch2!='\0')

{

printAllPossibleCodes(input.substr(2), temp+ch2);

}

}

void printAllPossibleCodes(string input) {

/\*

Given the input as a string, print all its possible combinations. You do not need to return anything.

\*/

string temp;

printAllPossibleCodes(input, temp);

}

**Return Permutations - String**

**Send Feedback**

Given a string S, find and return all the possible permutations of the input string.

**Note 1 : The order of permutations is not important.**

**Note 2 : If original string contains duplicate characters, permutations will also be duplicates.**

**Input Format :**

String S

**Output Format :**

All permutations (in different lines)

**Sample Input :**

abc

**Sample Output :**

abc

acb

bac

bca

cab

cba

Sol

#include <string>

using namespace std;

int returnPermutations(string input, int fixed, string output[]){

if(fixed==input.length())

{

output[0]="";

return 1;

}

int size=returnPermutations(input, fixed+1, output);

int top=size;

for(int outer=0;outer<size;outer++)

{

string temp=output[outer];

output[outer]=input[fixed]+output[outer];

// cout<<"\*outer\*"<<output[outer]<<"\*"<<endl;

for(int inner=1;inner<=temp.length();inner++)

{

output[top]=temp.substr(0,inner)+input[fixed]+temp.substr(inner);

// cout<<"\*"<<output[top]<<"\*"<<endl;

top++;

}

}

return top;

}

int returnPermutations(string input, string output[])

{

return returnPermutations(input, 0, output);

}

Solution 2:

#include <string>

using namespace std;

int returnPermutations(string input, int fixed, string output[]){

/\* Don't write main() function.

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

\* Print output as specified in the question

\*/

static int count=0;

if(input.length()==fixed){

output[count] = input;

return count++;

}

for(int i=fixed;i<input.length();i++){

swap(input[i],input[fixed]);

returnPermutations(input,fixed+1,output);

swap(input[i],input[fixed]);

}

return count;

}

int returnPermutations(string input, string output[]){

/\* Don't write main() function.

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

\* Print output as specified in the question

\*/

return returnPermutations(input,0,output);

}

**Print Permutations - String**

**Send Feedback**

Given a string, find and print all the possible permutations of the input string.

**Note : The order of permutations are not important. Just print them in different lines.**

**Sample Input :**

abc

**Sample Output :**

abc

acb

bac

bca

cab

cba

Sol

#include <iostream>

#include <string>

using namespace std;

void printPermutations(int fixed,string input)

{

if(input.length()==fixed){

cout<<input<<endl;

return;}

int l=input.length();

for(int i=fixed;i<l;i++)

{

swap(input[fixed],input[i]);

printPermutations(fixed+1,input);

swap(input[fixed],input[i]);

}

}

void printPermutations(string input)

{

/\* Don't write main() function.

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

\* Print output as specified in the question

\*/

printPermutations(0,input);

}