Universidade Federal de Goiás
Instituto de Informática
Introdução à Programação - 2020-1
Lista de Exercícios L3 - Strings

Prof. Celso Gonçalves Camilo Júnior
Prof. Edmundo Sergio Spoto
Prof. Gilmar Ferreira Arantes
Prof. Gustavo Teodoro Laureano
Prof. Thierson Couto Rosa
Prof. Vagner José do Sacramento Rodrigues

Sumário

1 Combinador

 (+)

Implemente um programa denominado combinador, que recebe duas strings e deve combiná-las, alternando as letras de cada string, começando com a primeira letra da primeira string, seguido pela primeira letra da segunda string, em seguida pela segunda letra da primeira string, e assim sucessivamente. As letras restantes da cadeia mais longa devem ser adicionadas ao fim da string resultante e retornada.

Entrada

A entrada contém vários casos de teste. A primeira linha contém um inteiro *N* que indica a quantidade de casos de teste que vem a seguir. Cada caso de teste é composto por uma linha que contém duas cadeias de caracteres. Cada cadeia de caracteres contém entre 1 e 50 caracteres inclusive.

Saída

Combine as duas cadeias de caracteres da entrada como mostrado no exemplo abaixo e exiba a cadeia resultante.

Exemplo

Entrada

`2`
`Tpo oCder`
`aa bb`

Saída
`TopCoder` `abab`

2 LED

 (+)

João quer montar um painel de leds contendo diversos números. Ele não possui muitos leds, e não tem certeza se conseguirá montar o número desejado. Considerando a configuração dos leds dos números abaixo, faça um algoritmo que ajude João a descobrir a quantidade de leds necessário para montar o valor.

Entrada

A entrada contém um inteiro $N, (1 \leq N \leq 1.000)$ correspondente ao número de casos de teste, seguido de N linhas, cada linha contendo um número $(1 \leq V \leq 10^{100})$ correspondente ao valor que João quer montar com os leds.

Saída

Para cada caso de teste, imprima uma linha contendo o número de leds que João precisa para montar o valor desejado, seguido da palavra "leds".

Exemplo

Entrada
3 115380 2819311 23456
Saída
27 leds 29 leds 25 leds

3 Quantas Letras?

 (+)

Tia Magnólia está ensinando as crianças a reconhecerem letras, e entre as letras quais são vogais e quais são consoantes. Ela precisa fazer vários testes com seus alunos. Ela quer que eles leiam várias linhas de um texto e contem em cada linha quantas letras (maiúsculas ou minúsculas), quantas vogais (maiúsculas ou minúsculas) e quantas consoantes (minúsculas ou maiúsculas) existem em cada linha lida. Como Tia Magnólia possui vários textos, ela gostaria de uma forma automatizada de obter essa contagem para gerar um gabarito que permita a ela verificar se as respostas dos alunos estão corretas ou não. Sabendo que você é "FERA" em processamento de strings, ela quer que você faça um programa que gere essas contagens para ela.

Entrada

A entrada contém vários casos de teste. A primeira linha contém um inteiro N que indica a quantidade de casos de teste. Cada caso de teste consiste de uma única linha de texto. A linha pode conter qualquer tipo de caractere (letras e "não letras"). Uma linha pode conter até 10.000 caracteres.

Saída

Para cada caso de teste, imprima três mensagens, cada uma em uma linha diferente. A primeira mensagem deve estar no seguinte formato: "Letras = x". A segunda mensagem deve ser : "Vogais = y" e a última mensagem deve ser: "Consoantes = z". Os valores de x, y e z nas mensagens correspondem aos totais de, respectivamente, letras, vogais e consoantes encontrados em um caso de teste.

Exemplo

Entrada

`4`
`Este e um caso de teste dos varios possiveis`
`Vem ver vovo!`
`O presidente renunciou?`
`#chapeuzinho#Vermelho#`

Saída

`Letras = 36`
`Vogais = 17`
`Consoantes = 19`
`Letras = 10`
`Vogais = 4`
`Consoantes = 6`
`Letras = 20`
`Vogais = 10`
`Consoantes = 10`
`Letras = 19`
`Vogais = 8`
`Consoantes = 11`

4 Um_Dois_Três

 (+)

Seu irmão mais novo aprendeu a escrever apenas um, dois e três, em Inglês. Ele escreveu muitas dessas palavras em um papel e a sua tarefa é reconhecê-las. Nota-se que o seu irmão mais novo é apenas uma criança, então ele pode fazer pequenos erros: para cada palavra, pode haver, no máximo, uma letra errada. O comprimento de palavra é sempre correto. É garantido que cada palavra que ele escreveu é em letras minúsculas, e cada palavra que ele escreveu tem uma interpretação única.

Entrada

A primeira linha contém o número de palavras que o seu irmão mais novo escreveu. Cada uma das linhas seguintes contém uma única palavra com todas as letras em minúsculo. As palavras satisfazem as restrições acima: no máximo uma letra poderia estar errada, mas o comprimento da palavra está sempre correto. Haverá, no máximo, 1000 palavras de entrada.

Saída

Para cada caso de teste, imprima o valor numérico da palavra

Exemplo

Entrada
3 owe too theee
Saída
1 2 3

5 Zero Vale Zero

 (+)

Um dia o Prof. Humberto José Roberto fez o seguinte questionamento: Se o zero a esquerda de um número não tem valor algum, por que teria em outras posições de um número? Analisando da seguinte forma, ele pede sua ajuda para, ao somar dois valores inteiros, que o resultado seja exibido segundo o raciocínio dele, ou seja, sem os Zeros. Por exemplo, ao somar 15 + 5, o resultado seria 20, mas com esta nova ideia, o novo resultado seria 2, e, ao somar 99 + 6, o resultado seria 105, mas com esta nova ideia, o novo resultado seria 15.

Escreva um programa que, dado dois números inteiros, sem o algarismo zero, some os mesmos e, caso o resultado tenha algum algarismo zero, que os retire antes de exibir.

Entrada

Haverá diversos casos de teste. Cada caso de teste inicia com dois inteiros M e N ($1 \leq M \leq N \leq 999.999.999$). O último caso de teste é indicado quando $N = M = 0$, sendo que este caso não deve ser processado.

Saída

Para cada caso de teste, imprima o resultado da soma dos dois valores, sem os zeros.

Sugestão

Ao somar os dois números utilize a função sprintf() para armazenar a soma em uma string.

Exemplo

Entrada
7 8 15 5 99 6 0 0
Saída
15 2 15

6 Palíndromo

 (++)

Um palíndromo é uma palavra, frase ou qualquer outra sequência de unidades que tenha a propriedade de poder ser lida tanto da direita para a esquerda como da esquerda para a direita. Em um palíndromo, normalmente são desconsiderados os sinais ortográficos (diacríticos ou de pontuação), assim como o espaços entre palavras.

A palavra "palíndromo" vem das palavras gregas palin ("para trás") e dromos ("corrida, pista"). Rômulo Marinho, veterano palindromista brasileiro, classifica os palíndromos em:

- Expliciti - trazem sempre uma mensagem direta, clara e inteligível, como "Socorram-me, subi no ônibus em Marrocos" (palíndromo de autoria anônima, provavelmente a mais conhecido em língua portuguesa).
- Interpretabiles - têm coerência, mas requerem esforço intelectual do leitor para serem entendidos, como "A Rita, sobre vovô, verbos atira".
- Insensati - cuidam apenas de juntar letras ou palavras sem se preocupar com o sentido, como "Olé! Maracujá, caju, caramelo".

As frases formando um palíndromo também são chamadas de anacíclicas, do grego anakúklein, significando que volta em sentido inverso, que refaz inversamente o ciclo.

Faça um programa em que receba uma sequencia de palavras e indique se ela é ou não um palíndromo.

Entrada

A entrada consiste de várias palavras, uma por linha com comprimento máximo de 200 caracteres por palavra. A entrada termina com EOF.

Saída

A saída consiste de várias linhas, uma para cada palavra. Você deve imprimir a palavra "sim", caso a palavra seja palindromo; ou "nao", caso contrário. Nos dois casos, em minúsculo, sem acento e sem as aspas duplas. Após a última resposta dada, quebre uma linha.

Exemplo

Entrada:
mesa
estojo
asa
janela
salas
reter
ralar
osso
oco

Saída:
nao
nao
sim
nao
sim
sim
sim
sim
sim

7 Prefixo de Uma String

Escreva um programa para ler várias linhas na entrada. Cada linha contém um número inteiro seguido por um espaço e por uma string. Para cada linha, o programa deve chamar uma função do tipo **ponteiro para char** que que receba como primeiro parâmetro *n* o inteiro lido e como segundo parâmetro *s* a string lida. A função deve alocar espaço suficiente para armazenar os *n* primeiros caracteres de *s* (prefixo de *s*). Deve copiar os *n* primeiros caracteres de *s* para essa nova string e retornar o endereço da string criada. Se *n* for maior que o tamanho da string *s*, o prefixo corresponde a uma cópia da string *s*. A função deve retornar NULL, se não conseguir alocar o espaço necessário para um prefixo. Após chamar a função, o programa deve verificar se função retornou um endereço válido de prefixo, e nesse caso, deve imprimir o prefixo e deve liberar a área ocupada pelo prefixo, antes de processar uma nova linha

Entrada

A primeira linha da entrada contém um inteiro positivo $N(1 \leq N \leq 20)$, o qual corresponde ao número de casos de teste. Cada caso de teste corresponde a uma linha. Cada linha possui um número inteiro positivo *n*, um espaço e uma string *s*, com no máximo 499 caracteres.

Saída

Para cada caso de teste o programa deve imprimir uma linha contendo o prefixo de tamanho *n* da string *s* lida naquele caso de teste.

Exemplo

Entrada:
5 1 Universidade Federal de Goias 0 Introducao a Programacao 3 Universidade Federal de Goias 20 Universidade Federal de Goias 30 Universidade Federal de Goias

Saída:
U Uni Universidade Federal Universidade Federal de Goias

8 Procura Caractere

Escreva um programa para ler várias linhas na entrada. Cada linha contém um caractere seguido por um espaço e por uma string. Para cada linha, o programa deve chamar uma função do tipo int que receba como primeiro parâmetro o caractere lido e como segundo parâmetro a string lida. A função deve retorna o índice do vetor onde o caractere aparece pela primeira vez na string. Se o caractere não aparece na string, a função deve retornar -1.

Entrada

A primeira linha da entrada contém um inteiro positivo $N(1 \leq N \leq 20)$, o qual corresponde ao número de casos de teste. Cada caso de teste corresponde a uma linha. Cada linha possui um caractere, um espaço e uma string, com no máximo 499 caracteres.

Saída

Para cada caso de teste o programa deve imprimir uma das seguintes frases:

- "Caractere *c* encontrado no indice *i* da string.", ou
- "Caractere *c* nao encontrado."

Exemplo

Entrada:
4 o Introducao a Programacao G Universidade Federal de Goias ; Universidade Federal de Goias   Universidade Federal de Goias

Saída:
Caractere o encontrado no indice 4 da string. Caractere G encontrado no indice 24 da string. Caractere ; nao encontrado. Caractere encontrado no indice 12 da string.

Observação: Na última linha de entrada do exemplo, o caractere a ser procurado é o caractere espaço.

9 Sequência Espelho

 (++)

Imprimir números em sequência é uma tarefa relativamente simples. Mas, e quando se trata de uma sequência espelho? Trata-se de uma sequência que possui um número de início e um número de fim, e todos os números entre estes, inclusive estes, são dispostos em uma sequência crescente, sem espaços e, em seguida, esta sequência é projetada de forma invertida, como um reflexo no espelho. Por exemplo, se a sequência for de 7 a 12, o resultado ficaria 789101112211101987.

Entrada

A entrada possui um valor inteiro C indicando a quantidade de casos de teste. Em seguida, cada caso apresenta dois valores inteiros, B e E $(1 \leq B \leq E \leq 12221)$, indicando o início e o fim da sequência.

Saída

Para cada caso de teste, imprima a sequência espelho correspondente.

Sugestão

Utiliza a função sprintf() para imprimir um número inteiro em uma string. Use a função strlen() para obter o tamanho de uma string.

Exemplo

Entrada
3 1 5 10 13 98 101
Saída
1234554321 1011121331211101 98991001011010019989

10 Criptografia

 (++)

Solicitaram para que você construisse um programa simples de criptografia. Este programa deve possibilitar enviar mensagens codificadas sem que alguém consiga lê-las. O processo é muito simples. São feitas três passadas em todo o texto.

Na primeira passada, somente caracteres que sejam letras minúsculas e maiúsculas devem ser deslocadas 3 posições para a direita, segundo a tabela ASCII: letra 'a' deve virar letra 'd', letra 'y' deve virar caractere '|' e assim sucessivamente. Na segunda passada, a linha deverá ser invertida. Na terceira e última passada, todo e qualquer caractere a partir da metade em diante (truncada) devem ser deslocados uma posição para a esquerda na tabela ASCII. Neste caso, 'b' vira 'a' e 'a' vira '`'.

Por exemplo, se a entrada for "Texto #3", o primeiro processamento sobre esta entrada deverá produzir "Wh{wr #3". O resultado do segundo processamento inverte os caracteres e produz "3# rw{hW". Por último, com o deslocamento dos caracteres da metade em diante, o resultado final deve ser "3# rvzgV".

Entrada

A entrada contém vários casos de teste. A primeira linha de cada caso de teste contém um inteiro $N(1 \leq N \leq 10^4)$, indicando a quantidade de linhas que o problema deve tratar. As N linhas contém cada uma delas $M(1 \leq M \leq 10^3)$ caracteres.

Saída

Para cada entrada, deve-se apresentar a mensagem criptografada.

Exemplo

Entrada
4 Texto #3 abcABC1 vxpdylY .ph vv.xwfxo.fd
Saída
3# rvzgV 1FECedc ks. \n{frzx gi.r{hyz-xx

11 Lê *strings* (+++)

 (+++)

Essa questão é opcional.

"Chega!!! Estou farto dos 'problemas' do `scanf` com o '%s'. Preciso de uma função mais estável para usar em meus códigos" . Já é sabido que a função `scanf` para ler *strings* apresenta problemas por conta de caracteres residuais de outras leituras no terminal. Ao pressionarmos ENTER no terminal, são enviados dois caracteres, o "\r\n", com os respectivos códigos ASCII: 10, 13, onde '\n' indica o final de texto enviado pelo terminal. Por exemplo, ao digitarmos o texto "123"+ENTER no terminal, a sequência de caracteres passada é "123\r\n".

A função `scanf` processa o texto enviado pelo terminal de acordo com o código do formato passado como parâmetro. O '\n' é consumido e o texto residual "123\r" é processado pelo `scanf`. Ao processar o texto "123\r"com o código de formado "%d", a função `scanf` busca de uma sequência de caracteres da esquerda para a direita que representa um número inteiro até encontrar um caracter que não seja um dígito. O processamento resulta no número 123 e sobra o caracter '\r', que irá compor o próximo texto a ser lido no terminal.

Ao digitar um novo texto, por exemplo, "abc"+ENTER, o texto enviado pelo terminal torna-se "\rabc\r\n". Aí começam os problemas... Ao ler um número inteiro ou real, o '\r' é consumido e ignorado, mas ao ler um caracter ou uma *string* o '\r' é considerado. Além disso, ele indica fim de texto para a leitura de *strings*. Para a leitura de caracteres e *strings*, as expressões "%c", "%s"e "%[^\n]"precisam lidar com o problema do '\r'. O código abaixo, por exemplo, ao receber o texto "88"+ENTER via terminal, termina com a impressão "n: 88, c: 10", sem esperar pela leitura do caracter `ch`.

```
int n;
char ch;
scanf("%d", &n);
scanf("%c", &ch);
printf("n: %d, c: %d\n", n, ch);
```

Note que uma ligeira alteração no texto de entrada resolve esse problema para `ch`. Se informarmos o texto "88a"+ENTER via terminal, o código termina com a impressão "n: 88, c: 97", isso porque passamos o caracter 'a' antes do '\r'. O '\r' será um problema para a próxima leitura de um caracter ou *string*.

Uma alternativa a esse problema é incluir uma leitura de caracter adicional sempre que for necessário ler um caracter ou uma *string*, mas essa não é uma alternativa elegante.

Outra limitação do `scanf` é a característica de não ler espaços no início de uma *string*, o que pode ser um problema dependendo da aplicação que se deseja implementar. Além disso, não é seguro ler uma *string* uma vez que a função não sabe a quantidade de caracteres que o vetor pode armazenar.

Na verdade, as características do `scanf` citadas neste texto como "problemas"são comportamentos propositalmente implementados, que podem ser configurados para resolver esses e outros problemas mais complexos na leitura de *strings*.

Em vez de aprender a usar as expressões entendidas pelo `scanf`, faça uma função que leia *strings* de modo a não sofrer com problemas de caracteres residuais e que também permita a leitura de espaços. Adicionalmente, esse função deve retornar a quantidade de caracteres lidos e limitar a quantidade de carateres a serem lidos. Os caracteres "\r\n"não podem compor a *string* final. Lembre-se que uma *string* precisa do '\0' para indicar seu final. O protótipo da função deve ser o seguinte:

```
/**
```

```
 * @param str vetor de caracteres onde a string lida será gravada
 * @param n quantidade máxima de caracteres a ser lidos
 * @return quandidade de caracteres lidos
 */
int le_string( char * str, int n );
```

Seu programa principal deve ser o listado abaixo. A função `print_codes` imprime os códigos ASCII de cada caracter da *string* passada via parâmetro. Seu protótipo é:

```
/**
 * @param str string de entrada
 */
void print_codes( char * str );
```

Seu programa principal deve ser o listado abaixo. Esse código avalia se a função `le_string` consegue lidar com o caractere '\r'.

```
#define N 128+1

int main() {
    char str[N], s[N];
    char c;
    int i;

    scanf("%c", &c);
    le_string(str, 3);
    print_codes(str);
    printf("caracter:%c, str:%s\n", c, str);

    scanf("%c", &c);
    le_string(str, 5);
    print_codes(str);
    printf("caracter:%c, str:%s\n", c, str);

    scanf("%c", &c);
    le_string(str, 5);
    print_codes(str);
    printf("caracter:%c, str:%s\n", c, str);

    scanf("%d", &i);
    le_string(str, 3);
    print_codes(str);
    printf("inteiro:%d, str:%s\n", i, str);

    //printf("Digite inteiros separados por espaco: ");
    scanf("%d", &i);
    //printf("inteiro:%d\n", i);
    //printf("Le string (15):\n");
    le_string(str, 15);
    print_codes(str);
    printf("inteiro:%d, str:%s\n", i, str);

    //printf("Digite uma string sem espacos: ");
    scanf("%s", s);
    //printf("string:%s\n", str);
    //printf("Le string (10):\n");
    le_string(str, 100);
    print_codes(str);
```

```
42      printf("string:%s, str:%s\n", s, str);
43
44      //printf("Digite uma string com espacos: ");
45      scanf("%s", s);
46      //printf("string:%s\n", str);
47      //printf("Le string (20):\n");
48      le_string(str, 100);
49      print_codes(str);
50      printf("string:%s, str:%s\n", s, str);
51
52      return 0;
53  }
```

Entrada

- Um caracter + ENTER
- Texto com espaços + ENTER
- Sequência de caracteres sem espaços + ENTER
- Sequência de caracteres separados por espaços + ENTER
- Um número inteiro + ENTER
- Texto com espaços + ENTER
- Inteiros separados por espaços + ENTER
- Texto sem espaços + ENTER
- Texto com espaços + ENTER
- Texto com espaços + ENTER

Saída

Uma linha com os códigos de cada caractere da `str` lida pela função `le_string` seguida por uma linha com a apresentação do conteúdo do dado lido e da *string* `str`.

Observações

DICA: Você pode fazer uma função que leia caracter por caracter usando o "%c". Lembre-se que o texto passado pelo terminal é uma sequência de caracteres.

Exemplo

Entrada	Saída
x texto 123 abcdef x y z h w 99 texto com espacos 11 22 33 44 texto_sem_espacos texto2 com espacos Text with spaces	116,101,120 caracter:x, str:tex 98,99,100,101,102 caracter:a, str:bcdef 32,121,32,122,32 caracter:x, str: y z 116,101,120 inteiro:99, str:tex 32,50,50,32,51,51,32,52,52 inteiro:11, str: 22 33 44 116,101,120,116,111,50,32,99,111,109,32,101,115,112,97,99 string:texto_sem_espacos, str:texto2 com espacos 32,119,105,116,104,32,115,112,97,99,101,115 string:Text, str: with spaces

12 Aliteração

 (+++)

Uma aliteração ocorre quando duas ou mais palavras consecutivas de um texto possuem a mesma letra inicial (ignorando maiúsculas e minúsculas). Sua tarefa é desenvolver um programa que identifique, a partir de uma sequência de palavras, o número de aliterações que essa sequência possui.

Entrada

A entrada contém diversos casos de testes. Cada caso é expresso como um texto em uma única linha, contendo de 1 a 100 palavras separadas por um único espaço, cada palavra tendo de 1 a 50 letras minúsculas ou maiúsculas ('A'-'Z','a'-'z'). A entrada termina em EOF.

Saída

Para cada caso de teste imprima o número de aliterações existentes no texto informado, conforme exemplos abaixo.

Exemplo

Entrada

`He has four fanatic fantastic fans`
`There may be no alliteration in a sequence`
`Round the rugged rock the ragged rascal ran`
`area artic Soul Silly subway ant artic none`

Saída

`2`
`0`
`2`
`3`

13 Avance as Letras

 (+++)

São dadas na entrada uma string A e outra B. Em uma operação você pode escolher uma letra da primeira string e avançar esta letra. Avançar uma letra significa transformá-la na próxima letra do alfabeto, veja que a próxima letra depois de z vem a letra a novamente!

Por exemplo, podemos transformar a string **ab** em **bd** em no mínimo 3 operações: **ab** $\rightarrow$ **bb** $\rightarrow$ **bc** $\rightarrow$ **bd**. Podemos aplicar operações nas letras em qualquer ordem, outra possibilidade seria: **ab** $\rightarrow$ **ac** $\rightarrow$ **bc** $\rightarrow$ **bd**.

Dadas as duas strings, calcule o mínimo número de operações necessárias para transformar a primeira na segunda.

Entrada

Na primeira linha terá um inteiro $T(T \leq 100)$ indicando o número de casos de teste. Para cada caso, na única linha teremos as duas strings $A(1 \leq |A| \leq 10^4$ - sendo que $|A|$ significa o tamanho da string $A)$ e $B(|B| = |A|)$ separadas por um espaço. Ambas as strings são compostas apenas por letras minúsculas do alfabeto e são do mesmo tamanho.

Saída

Para cada caso imprima o número mínimo de operações.

Exemplo

Entrada
3 ab bd abc abc abcdefghiz aaaaaaaaaa
Saída
3 0 173

14 Sentença Dançante

 (+++)

Uma sentença é chamada de dançante se sua primeira letra for maiúscula e cada letra subsequente for o oposto da letra anterior. Espaços devem ser ignorados ao determinar o case (minúsculo/maiúsculo) de uma letra. Por exemplo, "A b Cd"é uma sentença dançante porque a primeira letra (’A’) é maiúscula, a próxima letra (’b’) é minúscula, a próxima letra (’C’) é maiúscula, e a próxima letra (’d’) é minúscula.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é composto por uma linha que contém uma sentença, que é uma string que contém entre 1 e 50 caracteres (’A’-’Z’,’a’-’z’ ou espaço ’ ’), inclusive, ou no mínimo uma letra (’A’-’Z’,’a’-’z’). A entrada termina por fim de arquivo.

Saída

Transforme a sentença de entrada em uma sentença dançante (conforme o exemplo abaixo) trocando as letras para minúscula ou maiúscula onde for necessário. Todos os espaços da sentença original deverão ser preservados, ou seja, "sentence "deverá ser convertido para "SeNtEnCe ".

Exemplo

Entrada

`This is a dancing sentence`
`  This is a dancing sentence`
`aaaaaaaaaa`
`z`

Saída

`ThIs Is A dAnCiNg SeNtEnCe`
`  ThIs Is A dAnCiNg SeNtEnCe`
`AaAaAaAaAaA`
`Z`

15 Frequência de Letras

 (+++)

Neste problema estamos interessados na frequência das letras em uma dada linha de texto. Especificamente, deseja-se saber qual(is) a(s) letra(s) de maior frequência do texto, ignorando o "case sensitive", ou seja maiúsculas ou minúsculas (sendo mais claro, "letras" referem-se precisamente às 26 letras do alfabeto).

Entrada

A entrada contém vários casos de teste. A primeira linha contém um inteiro N que indica a quantidade de casos de teste. Cada caso de teste consiste de uma única linha de texto. A linha pode conter caracteres "não letras", mas é garantido que tenha ao menos uma letra e que tenha no máximo 200 caracteres no total.

Saída

Para cada caso de teste, imprima uma linha contendo a(s) letra(s) que mais ocorreu(ocorreram) no texto em minúsculas (se houver empate, imprima as letras em ordem alfabética).

Exemplo

Entrada

`3`
`Computers account for only 5% of the country's commercial electricity consumption.`
`Input`
`frequency letters`

Saída

`co`
`inptu`
`e`

16 Limpa *String* (+++)

 (+++)

Faça um programa que atualize um texto removendo uma lista de caracteres indesejados. Tanto o texto quanto a lista de caracteres devem ser lidos no formato de *strings*.

Escreva a função `str_clean` que realiza o processamento desejado. Ela deve receber como parâmetros a *string* original `str` e a *string* com caracteres indesejados `clr`. Considere o tamanho máximo de 256 caracteres.

Sua função `str_clean` deve varrer a *string* original e remover todos os caracteres que ocorrem na *string* `clr`. Use um vetor de no máximo 256 caracteres. Seu programa principal deve ser o seguinte código:

```c
int main() {
    char str[N]; // string original
    char clr[N]; // lista de caracteres indesejados
    scanf("%[^\n]", str);
    scanf("\n%[^\n]", clr);
    str_clean(str, clr) ;
    printf("%s\n", str);
    return 0;
}
```

Entrada

Seu programa deve ler duas *strings*.

Saída

Uma linha contendo a *string* modificada.

Observações

Exemplo

Entrada	Saída
`Fulando de Tal da Silva` `aeiou`	`Flnd d Tl d Slv`

Entrada	Saída
`100 200 300 400 500 600 700` `123456789`	`00 00 00 00 00 00 00`

Entrada	Saída
`1111111111x` `1`	`x`