P1: Find Words

Problem

找出英文句字中所有單字。

Input

輸入有多筆測資，每筆測資輸入一行字串 S，S 必須以 C++ string 儲存。

Output

每筆測資請輸出英文句字中所有單字，所有單字請按英文字母順序大到小排序，每筆測資輸出時，請以分行隔開。

Sample Input	Sample Output
Today is a sunny day.↵ You will go to school Monday.↵	sunny↵ is↵ day↵ a↵ Today↵ ↵ will↵ to↵ school↵ go↵ You↵ Monday↵

P2：信用卡號

Problem

銀行信用卡之卡號由四組長度為 4 的阿拉伯數字組合而成，四組數字間以字元-隔開，如 2617-2551-0219-2318 即為一組合法之信用卡號。

Input

輸入有多筆測資，每筆測資輸入一行字串 S，S 必須以 C++ string 儲存。

Output

字串 S 為合法之信用卡號時，輸出"TRUE"並印出四組數字相加之結果，否則，輸出"FALSE"。。

Sample Input	Sample Output
1111-2222-3333-0000↵ 1234-120A-0000-1234↵	TRUE 6666↵ ↵ FALSE↵

P3：電話號碼

Problem

假設電話號碼格式為：+(國碼)-區碼-電話號碼，國碼與區碼分別為長度為 3 及 1 的阿拉伯數字組合而成，電話號碼格式為 xddd-dddd，x 為區碼數字，d 為阿拉伯數字，如 +886-2-2905-3698 即為一組合法之電話號碼，為求資訊安全請將+886-2-2905-3698 之電話號碼換成+XXX-X-XXXX-3698。

Input

輸入測資有多筆字串 S，每行有一字串，字串中可以有空白字元(含\t 與\n)，S 必須以 C++ string 儲存。

Output

輸出原始字串 S 及加密後字串。

Sample Input

```
My phone number is +(886)-2-2905-3698.↲
Yours is +(886)-3-3322-0956↲
The wrong number is +(123)-5-4412-1233.↲
Today is a sunny day and I will call the number +(123)-1-1123-3455.↲
+(886)-3-333-4456↲
+(88)-2-2905-2698↲
+(aaa)-b-b123-3388↲
Goodbye!↲
```

Sample Output

```
My phone number is +(XXX)-X-XXXX-3698.↲
Yours is +(XXX)-X-XXXX-0956.↲
The wrong number is +(123)-5-4412-1233.↲
Today is a sunny day and I will call the number +(XXX)-X-XXXX-3455.
+(886)-3-333-4456↲
+(88)-2-2905-2698↲
+(aaa)-b-b123-3388↲
Goodbye!↲
```

P4：Floating Constant

Problem

請參考第三頁之 Grammars for C++ Expression，以 C++之 Regular Expression 描述 floating constant。

Input

輸入有多筆測資，每筆測資輸入一行字串 S，S 必須以 C++ string 儲存。

Output

字串 S 為 floating constant 時，輸出"TRUE"，否則，輸出"FALSE"並印出字串 S 中所有 floating constant 的位置及其內容，每筆測資輸出請以空白行隔開。

Sample Input	Sample Output
2.3e-12F↵ 12↵ 3e+3↵ A=12.3+10-.24*1.5e3↵	TRUE↵ ↵ FALSE↵ ↵ TRUE↵ ↵ FALSE↵ 2 12.3↵ 10 .24↵ 14 1.5e3↵

Grammar of C++ Expressions

Visual Studio .NET 2003

expression:

assignment-expression

expression , *assignment-expression*

assignment-expression:

conditional-expression

unary-expression assignment-operator assignment-expression

assignment-operator: one of

= *= /= %= += –= >= <= &= ^= |=

conditional-expression:

logical-or-expression

logical-or-expression **?** *expression* **:** *conditional-expression*

logical-or-expression:

logical-and-expression

logical-or-expression **||** *logical-and-expression*

logical-and-expression:

inclusive-or-expression

logical-and-expression **&&** *inclusive-or-expression*

inclusive-or-expression:

exclusive-or-expression

inclusive-or-expression **|** *exclusive-or-expression*

exclusive-or-expression:

and-expression

exclusive-or-expression **^** *and-expression*

and-expression:

equality-expression

and-expression **&** *equality-expression*

equality-expression:

relational-expression

equality-expression **==** *relational-expression*

equality-expression **!=** *relational-expression*

relational-expression:

shift-expression

relational-expression **<** *shift-expression*

relational-expression **>** *shift-expression*

relational-expression **<=** *shift-expression*

relational-expression **=>** *shift-expression*

shift-expression:

additive-expression

shift-expression **<<** *additive-expression*

shift-expression **>>** *additive-expression*

additive-expression:

multiplicative-expression

additive-expression **+** *multiplicative-expression*

additive-expression **–** *multiplicative-expression*

multiplicative-expression:

segment-expression

multiplicative-expression ***** *segment-expression*

multiplicative-expression **/** *segment-expression*

multiplicative-expression **%** *segment-expression*

segment-expression:

pm-expression

segment-expression **:>** *pm-expression*

pm-expression:

cast-expression

pm-expression **.*** *cast-expression*

pm-expression **–>*** *cast-expression*

cast-expression:

unary-expression

(*type-name* **)** *cast-expression*

unary-expression:

postfix-expression

++ *unary-expression*

— *unary-expression*

unary-operator cast-expression

sizeof *unary-expression*

sizeof (*type-name* **)**

allocation-expression

deallocation-expression

unary-operator: one of

*** & + – ! ~**

allocation-expression:

::$_{opt}$ **new** *placement*$_{opt}$ *new-type-name new-initializer*$_{opt}$

::$_{opt}$ **new** *placement*$_{opt}$ **(** *type-name* **)** *new-initializer*$_{opt}$

placement:

(*expression-list* **)**

new-type-name:

type-specifier-list new-declarator$_{opt}$

new-declarator:

ms-modifier-list $_{opt}$ ***** *cv-qualifier-list* $_{opt}$ *new-declarator*$_{opt}$

ms-modifier-list $_{opt}$ *complete-class-name* **::** ***cv-qualifier-list**$_{opt}$

new-declarator$_{opt}$

new-declarator$_{opt}$ **[** *expression* **]**

new-initializer:

(*initializer-list* **)**

deallocation-expression:

::$_{opt}$ **delete** *cast-expression*

::$_{opt}$ **delete [ ]** *cast-expression*

postfix-expression:

primary-expression

postfix-expression **[** *expression* **]**

postfix-expression **(** *expression-list* **)**

simple-type-name **(** *expression-list* **)**

postfix-expression **.** *name*

postfix-expression **–>** *name*

postfix-expression **++**

postfix-expression **—**

dynamic_cast < ***type-id* **> (*expression* **)**

static_cast < ***type-id* **> (*expression* **)**

const_cast < ***type-id* **> (*expression* **)**

reinterpret_cast < ***type-id* **> (*expression* **)**

typeid(*expression*)

typeid(*type-id*)

expression-list:

assignment-expression

expression-list **,** *assignment-expression*

primary-expression:

literal

this

:: *identifier*

:: *operator-function-name*

:: *qualified-name* **(** *expression* **)**

name

name:

identifier

operator-function-name

conversion-function-name

~ *class-name*

qualified-name

qualified-name:

ms-modifier-list$_{opt}$ *qualified-class-name* **::** *name*

literal:

integer-constant

character-constant

floating-constant

string-literal

integer-constant:

decimal-constant integer-suffix$_{opt}$

octal-constant integer-suffix$_{opt}$

hexadecimal-constant integer-suffix$_{opt}$

'*c-char-sequence***'**

decimal-constant:

nonzero-digit

decimal-constant digit

*octal-constan*t:

0

octal-constant octal-digit

hexadecimal-constant:

0x *hexadecimal-digit*

0X *hexadecimal-digit*

hexadecimal-constant hexadecimal-digit

nonzero-digit: one of

1 2 3 4 5 6 7 8 9

octal-digit: one of

0 1 2 3 4 5 6 7

hexadecimal-digit: one of

0 1 2 3 4 5 6 7 8 9

a b c d e f

A B C D E F

integer-suffix:

unsigned-suffix long-suffix$_{opt}$

long-suffix unsigned-suffix$_{opt}$

unsigned-suffix: one of

u U

long-suffix: one of

l L

character-constant:

'*c-char-sequence***'**

L'*c-char-sequence***'**

c-char-sequence:

c-char

c-char-sequence c-char

c-char:

any member of the source character set except the single quote ('), backslash (\ ), or newline character

escape-sequence

escape-sequence:

simple-escape-sequence

octal-escape-sequence

hexadecimal-escape-sequence

simple-escape-sequence: one of

**\' \" \? **

\a \b \f \n \r \t \v

octal-escape-sequence:

\ *octal-digit*

\ *octal-digit octal-digit*

\ *octal-digit octal-digit octal-digit*

hexadecimal-escape-sequence:

\x*hexadecimal-digit*

hexadecimal-escape-sequence hexadecimal-digit

floating-constant:

fractional-constant exponent-part$_{opt}$ *floating-suffix*$_{opt}$

digit-sequence exponent-part floating-suffix$_{opt}$

fractional-constant:

digit-sequence$_{opt}$ **.** *digit-sequence*

digit-sequence **.**

exponent-part:

e *sign*$_{opt}$ *digit-sequence*

E *sign*$_{opt}$ *digit-sequence*

sign: one of

+ –

digit-sequence:

digit

digit-sequence digit

floating-suffix: one of

f l F L

string literal:

"*s-char-sequence* $_{opt}$**"**

L "*s-char-sequence* $_{opt}$**"**

s-char-sequence:

s-char

s-char-sequence s-char

s-char:

any member of the source character set except double quotation marks ("), backslash (****), or newline character

escape-sequence