04　文字列

目標

□　`string` 型を使うことができるようになる.

□　文字と文字列の違いを理解する.

□　文字列の各文字に対して処理が行えるようになる.

例題

【キャピタリゼーション】

出典：JOI 2019/2020 一次予選（第 3 回）B

問題文

JOI 君は長さ N の文字列 S を見つけた．S に含まれる文字はすべて英小文字である.

JOI 君はこの文字列から自分の名前である `joi` が連続している部分を先頭から順にすべて探しだすことにした. そして `joi` を見つけるたびに, 強調のためにそれを大文字の `JOI` に置き換えることにした.

文字列 S が与えられたとき，S に含まれる `joi` をすべて `JOI` に置き換えた文字列を出力するプログラムを作成せよ.

制約

- $3 \leqq N \leqq 100$.
- S は長さ N の文字列である.
- S は英小文字からなる.

入力

入力は以下の形式で標準入力から与えられる.

```
N
S
```

出力

S に含まれる `joi` をすべて `JOI` に置き換えた文字列を 1 行で出力せよ.

入力例 1

```
11
joinojoijin
```

出力例 1

```
JOInoJOIjin
```

1 文字目から 3 文字目の joi と 6 文字目から 8 文字目までの joi を JOI に置き換える.

入力例 2

```
16
jjooiiijoiojioij
```

出力例 2

```
jjooiiiJOIojioij
```

考察

- 文字列を前から順に 3 文字ずつ見ていきたい.
- for 文でカウンタ i を 1 から $N-2$ の範囲で変化させる.
- S_{i-1} が j で S_i が o で S_{i+1} が i ならば，それぞれ J，O，I に置き換える.

解答例

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int N;
    string S;
    cin >> N >> S;
    for(int i=1; i<N-1; i++){
        if(S[i-1] == 'j' && S[i] == 'o' && S[i+1] == 'i'){
            S[i-1] = 'J';
            S[i] = 'O';
            S[i+1] = 'I';
        }
    }
    cout << S << endl;
}
```

解説

① string 型

文字列を扱う変数の型は **string 型** である．文字列の代入や結合，辞書順の比較を行う際は，**ダブルクオーテーション " "** で囲む．

```
string S = "ABCXYZ";
```

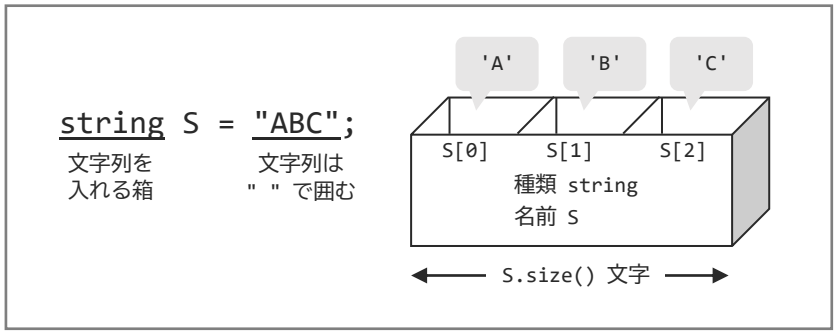
例えば上のように記述すると，変数 S には文字列 "ABCXYZ" が代入される．
また，S の長さは S.size() として求められる．文字列の結合は + 演算子で行える（表現 2 参照）．

② char 型

文字を扱う変数の型は **char 型** である（char は character（英: 文字）の略）．文字の代入や比較を行う際は，**シングルクオーテーション ' '** で囲む．
文字はただ 1 文字のことを指し，文字列は 0 個以上の文字が連なった列であることに注意する．

③ string 型と char 型の関係

文字列 S の i 文字目の文字は S[i] として求められる．ただし，0 文字目から始まり，最後は S.size()-1 文字目であることに注意する．



したがって，例えば次のように書ける．

```
char c = S[0];
```

これを踏まえて解答例を見てみよう．

```
for(int i=1; i<N-1; i++){
    if(S[i-1] == 'j' && S[i] == 'o' && S[i+1] == 'i'){
        S[i-1] = 'J';
        S[i] = 'O';
        S[i+1] = 'I';
    }
}
```

この場合，S の i-1 文字目，i 文字目，i+1 文字目を取り出して，比較や代入を行っている．文字列全体ではなく 1 文字ずつ着目して処理をしているため，シングルクオーテーションを用いている．

④ char 型の裏技

実は，char 型の正体は整数型であり，足し算や引き算がこれまで通り行える．0 から 127 までの整数を次の表のように各文字と対応させることで，文字を扱えるようにしているのである．

0	null	16	DLE	32	SP	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	TAB	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	¥	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

ここで，数字やアルファベットが連続していることに注意する．これにより，例えばある文字 c が英大文字であるかを次のように判定できる．

```
if('A' <= c && c <= 'Z'){

}
```

この場合，実際には文字と対応する数が 65 以上 90 以下か判定していることになる．
この性質を使うことで，特に大文字と小文字の区別が重要な問題を楽に解くことができる場合がある．

表現

(1) 標準入力から文字列 S を受け取り，S の末尾が s であるか判定する

```
string S;
cin >> S;
if(S[S.size()-1] == 's'){
    cout << "Yes" << endl;
}
else{
    cout << "No" << endl;
}
```

(2) 標準入力から文字列 S を受け取り，S を 2 つ繋げた文字列を出力する

```
string S;
cin >> S;
cout << S + S << endl;
```

(3) 標準入力から文字列 S を受け取り，S 中に含まれる J の個数を出力する

```
string S;
cin >> S;
int ans = 0;
for(int i=0; i<S.size(); i++){
    if(S[i] == 'J'){
        ans++;
    }
}
cout << ans << endl;
```

(4) 標準入力から自然数 N を受け取り，N の各桁の和を出力する

```
string N;
cin >> N;
int sum = 0;
for(int i=0; i<N.size(); i++){
    sum += N[i] - '0';
}
cout << sum << endl;
```

※ この実装は **解説 ④ char 型の裏技** を応用している.

(5) 標準入力から英小文字からなる文字列 S を受け取り，S の各文字をすべて大文字にして出力する

```
string S;
cin >> S;
for(int i =0; i<S.size(); i++){
    S[i] = S[i] - 'a' + 'A';
}
cout << S << endl;
```

※ この実装は **解説 ④ char 型の裏技** を応用している.

類題

- □ Plural Form（AtCoder Beginner Contest 179 A）
- □ IOI 文字列（JOI 2020/2021 一次予選（第 3 回）B）
- □ JOI ソート（JOI 2020/2021 一次予選（第 1 回）B）
- □ シーザー暗号（JOI 2006/2007 予選 B）
- □ 文字列の反転（JOI 2019/2020 一次予選（第 2 回）B）