Problem L. Programmable Virus

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3 seconds
Memory limit:	1024 megabytes

A great old ancient country is developing viruses for military purposes. Currently, they have found a way to do computation in human cells. They designed a special enzyme. This enzyme has 4 binding points. One is for reading RNAs as inputs, one is for generating RNAs as outputs. One is for both reading and generating RNA back and forth to keep temporary computation states, and the final one is for reading an RNA back and forth which indicates the program commands. They need some sample programs (RNAs) for demonstration. They don't have enough time to prepare the programs. So they ask for your help.

The enzyme deals with 3 nucleobases (G,U,A,C) as a unit when reading and writing RNAs. Not all $4^3 = 64$ combinations of nucleobases are in use. They only choose 11 of them. They interpret them as numeric data, or commands.

The 11 combinations are:

Nucleobases	Value	Command
GAC	-1	
CCC	0	STOP
ACG	1	NEXT
UGA	2	PREV
UGC	3	INC
UAC	4	DEC
GCG	5	OUT
UCC	6	IN
AGG	7	BEGIN
UGU	8	END
CAC	9	(DEBUG)

Once the input binding point of an enzyme is attached to an RNA chain, the computing process starts. The enzyme moves the input binding point to the first data unit of the input RNA, moves the program binding point to the first code point of the program RNA, initializes the output binding point to empty, and generates an RNA which contains only one unit and then initializes the state binding point to it.

The enzyme executes the program commands one by one, until reaching a command called 'STOP mark'. The program commands are very simple, much different from modern programming languages. There are only 9 kinds of commands:

0 STOP mark. Stop the execution.

1 Move the state binding point to the next one. If it is after the last one, put a 0 unit.

2 Move the state binding point to the previous one. If it is before the first one, put a 0 unit.

3 Replace the data unit under the state binding point with the numerically larger one. If it goes above the largest one (9), then replace it with -1.

4 Replace the data unit under the state binding point with the numerically smaller one. If it goes below the smallest one (-1), then replace it with 9.

5 Copy the data unit under the state binding point to the output point.

6 If there are no input data units, replace the data unit under the state binding point with -1. Otherwise, replace the data unit under the state binding point with the data unit under the input point. And move the input binding point to the next data unit.

7 If the data unit under the state binding point is not 0, then move the program bind point to the next command. Otherwise, move the program binding point forward to the command after the 'matching' 8 command.

8 If the data unit under the state binding point is 0, then move the program binding point to the next command. Otherwise, move the program binding point backward to the command after the 'matching' 7 command.

The word 'matching' means to treat command 7 and command 8 as parentheses '(' and ')'. They should form a valid paired expression. And the matched commands correspond to the matched parentheses. The paired command 7 and command 8 are often used as a loop.

The program binding point moves to the next command after executing the current one, except command 0,7, and 8.

The demonstration problem is simple: Read a series of non-negative digits as a decimal number and determine if it is a multiple of k. Additionally, the program length and the running steps should not exceed 10^6.

They prepared a simulation program (simu.c) for you. The first line of the input for the simulator is the RNA program. The following lines are input RNAs for the RNA program. Use the symbol '-' in the input to represent -1, and symbols '0' to '9' to represent 0 to 9. The simulator will execute the program for each input data, and output the final result and the running step counts. The simulator additionally accepts a special command, '9'. Every time it reaches command '9', it will output the current result, the state data, and the running step counts. The execution of command '9' didn't count into the steps.

Notes:

- You can download the source code of the simulator in the attachments of DOMjudge. You can find them here.
- The simulator that used to judge the submissions won't accept command '9'.

```c
/* simu.c */
/* gcc -o simu simu.c */
#include <stdio.h>
#include <string.h>

char * cmds[10] =
    { "CCC", "ACG", "UGA", "UGC", "UAC"
    , "GCG", "UCC", "AGG", "UGU", "CAC"
    };
char code[1000002], memory[2000001], input[1000001];
static inline int get_cmd(char * code){
    for(int i=9; i>=0; --i)
        if( strncmp(cmds[i], code, 3)==0 )
            return i;
    return -1;
}
int main(){
    if( !fgets(code, 1000002, stdin) ){
        puts("no code");
        return 1;
    }
    int code_len = strlen(code);
    if( code[code_len-1]!='\n' ){
        puts("code too long");
        return 1;
    }
    code[--code_len] = 0;

    while( fgets(input, 1000001, stdin) ){
        char * ip = code, * input_p = input;
        char *mem = memory + 1000000;
        char *mem_begin=mem, *mem_end=mem;
        *mem = 0;
        int step = 0, eof = 0;
        while(1){
            if( step == 1000000 ){
                puts(" step out of bound");
                break;
            }
            if( code+code_len-ip < 3 ){
                puts(" code out of bound");
                break;
            }
            int cmd = get_cmd(ip);
            if( cmd<0 ){
                printf(" invalid code point: %c%c%c\n",
                    ip[0], ip[1], ip[2]);
                break;
            }
            if( cmd==0 ){
                puts(" stop normally");
                break;
            }
```

```
if( cmd==1 ){
    ++mem;
    if( mem>mem_end )
        *++mem_end = 0;
}
else if( cmd==2 ){
    --mem;
    if( mem<mem_begin )
        *--mem_begin = 0;
}
else if( cmd==3 ){
    if( *mem==9 )
        *mem = -1;
    else
        ++*mem;
}
else if( cmd==4 ){
    if( *mem==-1 )
        *mem = 9;
    else
        --*mem;
}
else if( cmd==5 ){
    if( *mem==-1 )
        putchar('-');
    else
        putchar('0' + *mem);
}
else if( cmd==6 ){
    while(1){
        if( !*input_p || *input_p=='\n' ){
            *mem = -1;
            break;
        }
        if( *input_p=='-' ){
            *mem = -1;
            ++input_p;
            break;
        }
        if( '0'<=*input_p && *input_p<='9' ){
            *mem = *input_p++ - '0';
            break;
        }
        ++input_p;
    }
}
else if( cmd==7 ){
    if( !*mem ){
        int skip = 1;
        while(skip){
            ip += 3;
            if( code+code_len-ip < 3 ){
                puts(" 7 can't find matched 8");
```

```
                        goto END;
                    }
                    if( strncmp(cmds[8], ip, 3)==0 )
                        --skip;
                    else if( strncmp(cmds[7], ip, 3)==0 )
                        ++skip;
                }
            }
        }
        else if( cmd==8 ){
            if( *mem ){
                int skip = 1;
                while(skip){
                    ip -= 3;
                    if( ip < code ){
                        puts(" 8 can't find matched 7");
                        goto END;
                    }
                    if( strncmp(cmds[7], ip, 3)==0 )
                        --skip;
                    else if( strncmp(cmds[8], ip, 3)==0 )
                        ++skip;
                }
            }
        }
        else if( cmd==9 ){
            printf("\nstep: %d, state: ", step);
            for(char *p=mem_begin; p<=mem_end; ++p)
                if( p==mem )
                    printf("[%c]", *p<0 ? '-' : '0'+*p);
                else
                    putchar(*p<0 ? '-' : '0'+*p);
            printf(", code:");
            for(char *p=code; p<code+code_len; p+=3)
                printf("%c%c%c%c",
                    p==ip ? '[' : p==ip+3 ? ']' : ' ',
                    p[0], p[1], p[2]);
            if( code+code_len==ip+3 )
                putchar(']');
            puts("");
            --step;
        }
        ++step;
        ip += 3;
    }
    END:;
  }
  return 0;
}
```

Input

The only line includes a positive integer: k.

Constraints

- $1 \leq k \leq 6$
- $1 \leq n \leq 10^9$

Output

The output file should contain one line, which is the command RNA.

This program will read a series of non-negative digits as a decimal integer n. If n is a multiple of k, output 1, otherwise output 0.

Your program reads and writes numeric data directly. The enzyme will transform them from and to nucleobases.

The RNA code length should not exceed 10^6. The number of running steps should not exceed 10^6.

Examples

standard input	standard output
1	UCCAGGUACUGUUGCGCGCCC
2	UGCAGGUACACGAGGUACUGUUGAAGGUACAC GUGCUGAUGUUCCUGCUGUACGUGCAGGUACU GAUGCACGAGGUACUGAUACACGAGGUACUGA UGCACGAGGUACUGAUACACGAGGUACUGAUG CACGAGGUACUGAUACACGAGGUACUGAUGCA CGAGGUACUGAUACACGAGGUACUGAUGCACG AGGUACUGAUACACGUGUUGUUGUUGUUGUUG UUGUUGUUGUUGUUGAGCGCCC
3	UGCAGGACGACGUCCAGGUACACGUGCACGUG CUGAUGAUGUACGACGACGUGCUGAUGCAGGU ACACGAGGUACUGUUGAUGUACGAGGUGAUGA UGAUGCUGAAGGUACACGAGGUACUGUUGAUG UACGGCGCCCUGUUGAUGAAGGUACUGAUGCA CGAGGUACUGAUGCACGAGGUACUGAUACUAC ACGAGGUACUGAUGCACGAGGUACUGAUGCAC GAGGUACUGAUACUACACGAGGUACUGAUGCA CGAGGUACUGAUGCACGAGGUACUGAUACUAC ACGUGUUGUUGUUGUUGUUGUUGUUGUUGUUG AAGGUACUGAUGCACGUGUUGAAGGUACACGU GCUGAAGGUACACGUGCUGAAGGUACACGUAC UACUGAAGGUACACGUGCUGAUGUUGUUGUUG UACGAGGUACUGAUGCACGUGUUGAUGAUGU
6	UGCAGGACGACGUCCAGGUACACGUGCACGUG CUGAUGAUGUACGACGACGUGCUGAUGCAGGU ACACGAGGUACUGUUGAUGUACGAGGUGAUGA UGAUGAAGGAGGUACUGUGCGCCCUGUACGAC GACGACGACGACGUGCAGGUACUGAUGCACGA GGUACUGAUACACGAGGUACUGAUGCACGAGG UACUGAUACACGAGGUACUGAUGCACGAGGUA CUGAUACACGAGGUACUGAUGCACGAGGUACU GAUACACGAGGUACUGAUGCACGAGGUACUGA UACACGUGUUGUUGUUGUUGUUGUUGUUGUUG UUGUUGAGCGCCCUGUUGAACGACGAGGUACU GUACGAGGUACUGUUGAUGAUGAUGAAGGUAC ACGACGACGUGCACGUGCUGAUGAUGAUGAUG UACGACGACGAGGUACUGAUGAUGAUGCACGA CGACGUGUUGAUGAUGAAGGUACUGAUGCACG AGGUACUGAUGCACGAGGUACUGAUACUACAC GAGGUACUGAUGCACGAGGUACUGAUGCACGA GGUACUGAUACUACACGAGGUACUGAUGCACG AGGUACUGAUGCACGAGGUACUGAUACUACAC GUGUUGUUGUUGUUGUUGUUGUUGUUGAA GGUACUGAUGCACGUGUUGAAGGUACACGUGC UGAAGGUACACGUGCUGAAGGUACACGUACUA CUGAAGGUACACGUGCUGAUGUUGUUGUUGUA CGAGGUACUGAUGCACGUGUUGAUGAUGU

Notes

There's only one line in each sample output. We broke it into lines in samples because of the paper layout. You should concatenate them into one line.

The following is the running example of the first sample program.

Program	Memory	Input	Output	Comment
UCC AGG UAC UGU UGC GCG CCC		3		
UCC AGG UAC UGU UGC GCG CCC	3			IN
UCC AGG UAC UGU UGC GCG CCC	3			BEGIN: 3 != 0
UCC AGG UAC UGU UGC GCG CCC	2			DEC
UCC AGG UAC UGU UGC GCG CCC	2			END: 2 != 0
UCC AGG UAC UGU UGC GCG CCC	1			DEC
UCC AGG UAC UGU UGC GCG CCC	1			END: 1 != 0
UCC AGG UAC UGU UGC GCG CCC	0			DEC
UCC AGG UAC UGU UGC GCG CCC	0			END: 0 == 0
UCC AGG UAC UGU UGC GCG CCC	1			INC
UCC AGG UAC UGU UGC GCG CCC	1		1	OUT
UCC AGG UAC UGU UGC GCG CCC	1		1	STOP