記憶縛り (Limited Memory)

国際情報オリンピックの日本代表に選ばれた JOI ちゃんは，情報処理技術を高めるため，情報オリンピック日本委員会の K 理事長から以下のような課題を提示された．

K 理事長は，JOI ちゃんには分からないように，ある文字列 S を手帳に書いた．S は ‘<’, ‘>’, ‘[’, ‘]’ の 4 種類の文字からなる．K 理事長は JOI ちゃんに課題の内容と S の長さが書かれたプリントを渡した．

JOI ちゃんの課題は，S が**良い文字列**であるか否かを判定することである．ここで，良い文字列とは次のように定義される：

- 空文字列 (つまり，長さ 0 の文字列) は良い文字列である．
- x が良い文字列ならば，<x> (つまり，x を山括弧 <> で囲った文字列) は良い文字列である．
- x が良い文字列ならば，[x] (つまり，x を角括弧 [] で囲った文字列) は良い文字列である．
- x, y が良い文字列ならば，xy (つまり，x, y をこの順に連結した文字列) は良い文字列である．
- 以上で良い文字列と定義されるもののみが良い文字列である．

例えば，“<>[]” や “[<>]<>” は良い文字列であり，“><” や “[<]>” は良い文字列ではない．

JOI ちゃんは，毎日正午に高々 1 回，K 理事長に電話をすることができる．電話では，整数 I を指定することで，K 理事長から S の I 文字目を教えてもらうことができる．

さて，JOI ちゃんには，「この課題ではメモをとってはならない」という重大な制約が課されている．JOI ちゃんは毎晩 22 時に就寝し朝 6 時に起床するが，睡眠中に覚えていられるのは 22 ビットの情報のみである．より正確には，就寝前に 0 以上 $2^{22} - 1$ 以下の整数を 1 個記憶し，翌日は記憶した整数のみに基づいて課題に取り組まなければならない．ただし，S の長さはプリントに書いてあるので，いつでも参照することができる．

JOI ちゃんは，就寝前に整数を 1 個記憶する代わりに，S が良い文字列であるか否かを K 理事長に電子メールで回答してもよい．この場合課題は終了となり，正解か不正解かが判定される．ただし，課題の開始から 15 000 日以内に電子メールを送らなかった場合，不正解と判定される．

課題

JOI ちゃんの戦略を実装し，上記の課題で正解できるプログラムを作成せよ．

実装の詳細

あなたは，JOI ちゃんの戦略を実装した 1 個のプログラムを書かなければならない．プログラムは `Memory_lib.h` をインクルードすること．

プログラムは，以下の関数を実装しなければならない．

- `int Memory(int N, int M)`

  この関数は，S の長さおよび記憶した整数が与えられたときの，その日の JOI ちゃんの行動に対応する．

  - 引数 `N` は，文字列 S の長さ N である．
  - 引数 `M` は，前日の就寝前に記憶した内容を表す整数 M である．ただし，課題の開始時には $M = 0$ を記憶しているものとして扱う．
  - この関数の中では，以下で説明されるように関数 `Get` を高々 1 回呼び出すことができる．
  - この関数は，0 以上 $2^{22} - 1$ 以下の整数，または -1 または -2 を返さなければならない．この範囲の外の値を返した場合，**不正解 [1]** となる．
    - 戻り値が 0 以上 $2^{22} - 1$ 以下の整数の場合，就寝前にその値を記憶することを表す．
    - 戻り値が -1 の場合，「S は良い文字列である」と電子メールで回答することを表す．
    - 戻り値が -2 の場合，「S は良い文字列ではない」と電子メールで回答することを表す．
  - この関数は，引数 `N`, `M` の値および呼び出した `Get` の戻り値のみに依存した挙動を行うことが期待される．また，実際の採点プログラムではこの関数は $2^{22} \times 4$ 回呼び出される．詳細は，「採点の手順」と「重要な注意」を参照せよ．

プログラム中では以下の関数を呼び出すことができる．

- `char Get(int I)`

  この関数は，`Memory` の 1 回の呼び出しにつき高々 1 回しか呼び出すことができない．2 回以上呼び出した場合，**不正解 [2]** となる．

  引数 `I` は，$1 \leqq I \leqq N$ を満たす整数 I でなければならない．これを満たさない場合，**不正解 [3]** となる．

  この関数の戻り値は，文字列 S の I 文字目の文字を表す．

採点の手順

採点用の各入力データは複数のテストケースからなり，それらにおいて文字列 S の長さは同じ値 N である．

採点は以下の手順で行われる．不正解と判定された場合はその時点でプログラムは終了される．

(1) 引数 `N`, `M` の値および呼び出した `Get` の戻り値のすべての場合に対する `Memory` の挙動をまとめて調べる．すなわち，$0 \leqq M \leqq 2^{22} - 1$ を満たす整数 M それぞれについて，以下を行う：

(i) 文字 c を '<', '>', '[', ']' のそれぞれとして，以下を行う：

引数 `N` を N，引数 `M` を M として `Memory` を呼び出す．このとき，`Get` が呼び出されたならば，文字 c が返される．`Memory` が返した値を $m(M, c)$ とおく．

(ii) (i) での 4 回の `Memory` の呼び出しにおいて，`Get` を呼び出すか否かは同じでなければならない．さらに，`Get` を呼び出した場合は，4 回とも同じ整数 I を引数 `I` として呼び出さなければならず，`Get` を呼び出さなかった場合は，4 回とも同じ値を返さなければならない．これらを満たさない場合，**不正解 [4]** となる．`Get` を呼び出したときの引数 `I` を $i(M)$ とおく (ただし，`Get` を呼び出さなかった場合は $i(M) = 1$ とおく)．

(2) 各テストケースにおける文字列 S に対して，問題文で説明された課題のシミュレーションを行う．すなわち，以下を行う：

(i) $M = 0$ とする．

(ii) 以下を繰り返す：

(a) c を S の $i(M)$ 文字目の文字とする．

(b) M を $m(M, c)$ で置き換える．

(c) $M = -1$ または $M = -2$ ならば，(iii) へ進む．

(d) この項目に 15 000 回到達したら，**不正解 [5]** となる．

(iii) 以下のいずれかの場合，**不正解 [6]** となる：

- S が良い文字列だが，$M = -2$ である．
- S が良い文字列ではないが，$M = -1$ である．

(3) 正解となる．

重要な注意

- 実行時間計測・使用メモリ計測の対象となるのは，「採点の手順」における手順 (1) である．手順 (1) において `Memory` は $2^{22} \times 4$ 回呼び出されることに注意せよ．
- 手順 (1) における $2^{22} \times 4$ 回のすべての `Memory` の呼び出しにおいて，不正解 [1], [2], [3] となったり，実行時エラーを起こしたりしてはならないことに注意せよ．

コンパイル・実行の方法

作成したプログラムをテストするための，採点プログラムのサンプルが，コンテストサイトからダウンロードできるアーカイブの中に含まれている．このアーカイブには，提出しなければならないファイルのサンプルも含まれている．

採点プログラムのサンプルとして，grader-simple と grader-strict の 2 つが提供される．grader-simple のファイルは `grader-simple.c` または `grader-simple.cpp` であり，grader-strict のファイルは `grader-strict.c` または `grader-strict.cpp` である．例えば，作成したプログラムを `Memory.c` または `Memory.cpp` とするとき，作成したプログラムを grader-simple や grader-strict でテストするには，次のようにコマンドを実行する．

- C の場合

```
gcc -O2 -o grader-simple grader-simple.c Memory.c -lm
gcc -O2 -o grader-strict grader-strict.c Memory.c -lm
```

- C++ の場合

```
g++ -std=c++11 -O2 -o grader-simple grader-simple.cpp Memory.cpp
g++ -std=c++11 -O2 -o grader-strict grader-strict.cpp Memory.cpp
```

コンパイルが成功すれば，`grader-simple` あるいは `grader-strict` という実行ファイルが生成される．

実際の採点プログラムは，grader-simple あるいは grader-strict とは異なることに注意すること．grader-simple や grader-strict は単一のプロセスとして起動する．これらのプログラムは，標準入力から入力を読み込み，標準出力に結果を出力する．

採点プログラムのサンプルの概要

grader-simple は,「採点の手順」 (1) のように始めにまとめて `Memory` を呼び出すのではなく，あなたのプログラムと交互にやりとりを行うことによって問題文で説明された課題のシミュレーションを行う.「やりとりの例」を参照せよ．

grader-strict は,「採点の手順」と同様に `Memory` を呼び出す．

以下の点は実際の採点プログラムの挙動とは異なるので注意せよ：

- grader-simple は,「採点の手順」と `Memory` の呼び出し方が異なるので，不正解 [4] を判定しない．
- **grader-simple** や **grader-strict** は，不正解 **[6]** の判定を行わず，代わりに M の値を出力する．

採点プログラムのサンプルの入力

grader-simple や grader-strict は標準入力から以下の入力を読み込む.

- 1 行目には整数 N, Q が空白を区切りとして書かれている．N は文字列 S の長さを表し，Q ($0 \leqq Q \leqq 2^{31} - 1$) はテストケースの個数を表す.
- 続く Q 行の各行には，各テストケースにおける文字列 S (長さ N) が書かれている.

採点プログラムのサンプルの出力

grader-simple や grader-strict は標準出力へ以下の情報を出力する (引用符は実際には出力されない).

- 「採点の手順」(2)(iii) における M の値を (各テストケースごとに) 1 行に出力する.
- 不正解 [1], [2], [3], [4], [5] のいずれかと判定された場合，不正解の種類が “`Wrong Answer [1]`” のように出力される．その時点でプログラムは終了し，それ以降の出力は行われない.

制限

すべての入力データは以下の条件を満たす.

- $1 \leqq (S$ の長さ$) \leqq 100$.
- S の各文字は '<', '>', '[', ']' のいずれかである.

小課題

小課題 1 [10 点]

- $(S$ の長さ$) \leqq 8$ を満たす.

小課題 2 [10 点]

- $(S$ の長さ$) \leqq 14$ を満たす.

小課題 3 [5 点]

- $(S$ の長さ$) \leqq 24$ を満たす.

小課題 4 [5 点]

- $(S$ の長さ$) \leqq 30$ を満たす.

小課題 5 [10 点]

- S の各文字は '<', '>' のいずれかである.

小課題 6 [60 点]

追加の制限はない.

やりとりの例

grader-simple が読み込む入力の例と，それに対応する関数の呼び出しの例を以下に示す．

入力例	呼び出しの例			
	呼び出し	戻り値	呼び出し	戻り値
4 1 <>[]	Memory(4, 0)			
			Get(1)	
				<
		2015		
	Memory(4, 2015)			
			Get(3)	
				[
		3		
	Memory(4, 3)			
			Get(2)	
				>
		23		
	Memory(4, 23)			
			Get(4)	
				]
		4194303		
	Memory(4, 4194303)			
			Get(3)	
				[
		-1		

上のやりとりが行われたとき，grader-simple は −1 を出力する．