JOI shortcode 奮闘記

67 回生 小倉拳

1　あいさつ

どうもこんにちは。catupper こと小倉です。去年の 12 月に JOI（日本情報オリンピック）の予選がありました。情報オリンピックの問題は「このような問題を解いてくれるプログラムを組め」という形の問題です。具体的にどのようなものかは後を参照していただくとして、本オリンピックはプログラムを組みそれを回答として提出します。プログラム自体はちゃんと動くのであればどんな書き方でもいいのです。今回は shortcoding といって「ある機能をもつコードをできるだけ短い文字数で書く」というルールのもとそのプログラムを組んでいきたいと思います。これはその奮闘記です。

なお本記事での使用プログラミング言語は C です。またコンパイラは gcc4.5 とします。shortcode は基本的に一行のコードになるのですが見やすさのために適当な改行やインデントをしております。記事内の何 byte というのは 1 行にした時の値です。

2　shortcoding について

さて shortcode とはどのようなものでしょう。百聞は一見にしかず。具体例を見ていただきましょう。

```
b,o,s;main(a){for(;~scanf("%s",&a);)a>4e6?b=a%3?a
    &3?++o%3?b:(s=!printf("%d\n",b>3?b+s-3:s)):b+1:!(s
    -=~b):a;}
```

これは AOJ（Aizu Onlie Judge) の 0103 の問題の回答です。おそらく一番短い回答です。一見動かないように見えますが実際やってみるとちゃんと動きます。このコードに関して詳しく言うと時間がかかるので shortcoding の基本となる暗黙の了解なテクニックをまとめます。

1. #include<> は省く
   GCC は仕様として使用頻度の高い標準ライブラリを勝手に include してくれます。ただし include 文を一つも書かなかった場合に限ります。なんにせよ省けるのです。
2. 変数の型宣言は省く
   C の仕様として グローバル変数、関数、関数の引数の型が明示されていない場合 int 型と勝手に判断してくれます。故に変数はすべてグローバル変数もしくは関数の引数として作成します。
3. 変数の初期化は極力抑える
   こちらは GCC の仕様ではなくてプリプロセッサの問題なのですがメモリの確保の過程でグローバル変数を 0 にしてくれます。よって初期化しなくても 0 が入っています。これは配列でも同じなので memset(array,0,sizeof(array)) みたいなことをしなくても最初からすべて 0 で初期化されます。グローバル変数は 0 で初期化されますのでそれでいいのですが、関数の引数はそうは行きません。main 関数の引数ですが第一引数がコマンドライン引数の数、第二引数がコマンドライン引数を格納している配列のポインタがはいります。第二引数はめったに使いませんが、第一引数は基本的に 1 で初期化されます。よって 1 で初期化したい場合は main 関数の第一引数で変数を宣言するといいでしょう。
4. return 文は書かない
   今回の JOI は入力データが渡されてそれを処理したものを送信するだ

けなので 0 を返す必要がありません。ですが shortcoding というのは環境依存なテクニックを使えばどんどん短くなるということがありますので、競技プログラミングとして shortcoding をするときは一つの共通の環境を使用します。そのときの「共通の環境」としてよく使われるのが、オンラインジャッジや ideone.com です。先ほどの AOJ もオンラインジャッジです。共通の環境と共に問題の提供とその採点もしてくれるので競技 shortcoding では重宝します。ideone も共有しやすさからよく使われます。そこで return 文なのですが、オンラインジャッジでは不正なコードの対策としてちゃんと main 関数が最後に 0 を返すコードじゃないと実行してくれなかったりします。そのため shortcoding では最後に 0 を返すのが暗黙の了解となっております。ですのでこの記事において return 0 は欠かさず実装しようと思います。

さて、その return 文ですが、低層な部分で何をやってるかといいますと、CPU の問題になりますが return する値を eax レジスタに入れるという動作をしています。またグローバル変数に値を渡すとき、変数が値をもらうときに eax レジスタを通してもらいます。なので最後にグローバル変数に 0 を代入をすればあたかも return 0 されたのと同じ状態になります。これで return 文を書く手間を省きコードを短くします。この例では for 文の条件式の部分がループを脱すると同時に 0 を代入しているのでそこで return 0 されます。

5. コンマ演算子を使う

   コンマ演算子は左から順番に式を実行して行って最後の値を返します。とくにセミコロンで複数の式を書くのと変わりないのですが、なぜコンマを使うかというと、for 文や if 文の中身をコンマ演算子で一文にすることで中括弧を省略できるからです。これで 2byte 縮みます。

とりあえずこれらのテクニックは暗黙に使用しますのでご了承ください。

3　実際にやってみる。

さてそれでは実際に JOI 予選を解いていきましょう。問題文は一応載せておきますが、読みにくいと思われるので、元の問題が乗っている公式の URL を載せておきます。ググっても出てくると思います。

http://www.ioi-jp.org/joi/2011/2012-yo-prob_and_sol/index.html

一 問目

問題文

『JOI パスタ店では、ランチのおすすめパスタと搾りたてジュースのセットメニューが好評である。このセットメニューを注文するときは、その日の 3 種類のパスタと 2 種類のジュースから 1 つずつ選ぶ。パスタとジュースの値段の合計から 50 円を引いた金額が代金となる。ある日のパスタとジュースの値段が与えられたとき、その日のセットメニューの代金の最小値を求めるプログラムを作成せよ。』

入力

『入力は 5 行からなり，1 行に 1 つずつ正の整数が書かれている.

- 1 行目の整数は 1 つ目のパスタの値段である.
- 2 行目の整数は 2 つ目のパスタの値段である.
- 3 行目の整数は 3 つ目のパスタの値段である.
- 4 行目の整数は 1 つ目のジュースの値段である.
- 5 行目の整数は 2 つ目のジュースの値段である.

ただし，与えられる入力データにおいては全てのパスタとジュースの値段は 100 円以上 2000 円以下であることが保証されている.』

入力例	出力例
100	450
200	
300	
400	
500	

アルゴリズム

普通に数字をもらって最小値をとって足すだけの問題です。ただ厄介なことに、はじめは 3 つで次は 2 つなので繰り返して同じ動作をさせることはできません。ですので普通に解くだけになります。

普通のコード

```
b,c,d,e;main(a){
  scanf("%d%d%d%d%d",&a,&b,&c,&d,&e);
  printf("%d\n",a<(b>c?c:b)?a:(b>c?c:b)+(d>e?e:d
      )-50);
  b=0;
}
```

コメント

基本的に shotcoder は if 文を使えませんので、代わりに**三項演算子**を使います。最小値も三項演算子を使っています。

最短のコード

```
b,c,d,e;main(a){
  scanf("%d%d%d%d%d",&a,&b,&c,&d,&e);
  b=!printf("%d\n",a=d>e?e:d-50+fmin(a,b>c?c:b));
```

```
}
```

解説

b=!printf の部分は printf の**戻り値**が出力した文字数になることを利用し、b に 0 を代入しています。return の省略のテクニックです。shortcoding では各関数の戻り値を知ることが重要です。

fmin は 2 つの引数のうち小さい方を返す関数です。戻り値が double 型であることに注意してください。そうなると式の値が double にキャストされてしまいます。キャストされた状態で %d でフォーマットするとメモリサイズがあっていませんので間違った出力をしてしまいます。なので a=fmin... のように int 型の変数に代入しています。代入式の戻り値は代入した値ですので、ちゃんと int 型でフォーマットされます。

結果

99byte

Tips

三項演算子は欠かせないテクニックです。利点として if 文のように**括弧**を使わなくてよいという点や式の中にそのまま組み込める点が挙げられます。しかし括弧を使わずに、式の中に組み込むがゆえに、演算子の**優先順位**を気にしなくてはなりません。場合によっては期待しない動きをします。この問題も優先順位を気にしなかったらうまくいきません。こちらが失敗例

```
b,c,d,e;main(a){
 scanf("%d%d%d%d%d",&a,&b,&c,&d,&e);
 b=!printf("%d\n",a=fmin(a,b>c?c:b)+d>e?e:d-50);
}
```

違いは 3 行目。

fmin(a,b>c?c:b)+d>e?e:d-50

この部分です。何がいけないかというと > と + の優先順位の違いによって三項演算子の条件部が期待しないものになっております。優先順位的に

(fmin(a,b>c?c:b)+d>e)?e:d-50

というくくられ方をしてしまいます。本来は

fmin(a,b>c?c:b)+(d>e)?e:d-50

こうでなくてはならないので fmin の部分を左に持っていくことで解決します。

d>e?e:d-50+fmin(a,b>c?c:b)

優先順位には十分気を付けなければなりません。

二問目

問題文

『JOI 国ではサッカーが人気であり、JOI リーグというリーグ戦が毎週行われている．JOI リーグには N 個のチームが所属していて、1 から N までの番号がつけられている。すべての組み合わせの試合がちょうど一度ずつ行われる。つまり、$N \times (N-1)/2$ 試合が行われる。各試合の勝敗はそれぞれのチームの得点で決まる。勝ったチームの勝ち点は 3 点であり、負けたチームの勝ち点は 0 点である。引き分けの場合、両チームの勝ち点は 1 点である。順位は各チームの獲得した勝ち点の合計で決定し、得失点差は考えない。勝ち点の合計が等しいチームの順位は上位に揃える。全ての試合の結果が与えられたとき、各チームの順位を求めるプログラムを作成せよ。』

入力

『入力ファイルの 1 行目にはチームの個数 $N(2 \leq N \leq 100)$ が書かれている。続く $N \times (N-1)/2$ 行には各試合の結果が書かれている。$i+1$ 行目 $(1 \leq i \leq N \times (N-1)/2)$ には整数 $A_i, B_i, C_i, D_i (1 \leq A_i \leq N, 1 \leq B_i \leq N, 0 \leq C_i \leq 100, 0 \leq D_i \leq 100)$ が空白を区切りとして書かれており、チーム A_i とチーム B_i が対戦し、チーム A_i の得点が C_i 点、チーム B_i の得点が D_i 点であったことを

表す。全ての i について $A_i \neq B_i$ であり、同じ組み合わせの対戦が書かれていることはない。』

入力例	出力例
4	2
1 2 0 1	1
1 3 2 1	4
1 4 2 2	2
2 3 1 1	
2 4 3 0	
3 4 1 3	

アルゴリズム

各チームの点数は入力ごとに場合分けして加点していき、点数テーブルを作成。n 番目のチームの順位は自分よりも点数が大きいチームの数＋ 1 と等しいのでそれをループして数えます。

普通のコード

```
t[999],a,b,c,d,j;
main(i){
  for(scanf("%d",&i);~scanf("%d%d%d%d",&a,&b,&c,&d
      );){
       if(c!=d){
       if(c>d)t[a]+=3;
       else  t[b]+=3;
    }
    else{
      t[a]++;
```

```
        t[b]++;
      }
    }
    for(;j++-i;printf("%d\n",c+1))
      for(c=b=0;b++-i;)c+=t[j]<t[b];
}
```

解説

特に変なテクニックを使うようなところはありません。とりあえず、加点部分が冗長なのを何とかするために、**三項演算子**を使います。三項演算子は**添字の中にも使える**ので加点する配列の添字を指定することができます。

短いコード

```
t[999],a,b,c,d,j;
main(i){
  for(scanf("%d",&i);~scanf("%d%d%d%d",&a,&b,&c,&d
      );){
       c-d?t[c>d?a:b]+=3:t[a]++*++t[b]);
  }
  for(;j++-i;printf("%d\n",c+1))
    for(c=b=0;b++-i;)c+=t[j]<t[b];
}
```

解説

t[c>d?t:b] のように添字の中で三項演算子を使うことでなんとか一文にまとめています。また t[a]++*++t[b] というのは t[a]++ と ++t[b] を分けずに*で一つにまとめることで優先順位を高め、括弧がいらないようにしています。

しかしこれでは変数が多いのでひとつ減らします。よく見ると配列 t は t[0] が使われていないのでどれかの変数をこれで代用します。

最短のコード

```
t[999],b,c,d,j;
main(i){
   for(scanf("%d",&i);~scanf("%d%d%d%d",t,&b,&c,&d);){
        c-d?t[c>d?*t:b]+=3:t[*t]++*++t[b]);
   }
   for(;j++-i;c=!printf("%d\n",c))
      for(c=b=1;b+~i;)c+=t[j]<t[b++];
}
```

解説

これで一番短くなりました。配列の一番最初の要素は*t で参照できます。またそれのポインタは t なので scanf でアドレス演算子&を省略できます。for 文の中で b+ i という非常に怪しいことをやっていますがこの説明は第三問にとっておきましょう。

結果

171byte

三問目

問題文

『K 理事長は、JOI 市の中心部にある JOI ピザ店の常連客である。彼はある事情により、今月から節約生活を始めることにした。そこで彼は、JOI ピザ店で注文できるピザのうち、1 ドルあたりのカロリーが最大となるようなピザを注文したいと思った。このようなピザを「最高のピザ」と呼ぶことにしよう。「最高のピザ」は 1 種類とは限らない。JOI ピザでは N 種類のトッピングから何種類かを自由に選び、基本の生地の上に載せたものを注文することができ

る。同種のトッピングを 2 つ以上載せることはできない。生地にトッピングを 1 種類も載せないピザも注文できる。生地の値段は A ドルであり、トッピングの値段はいずれも B ドルである。ピザの値段は、生地の値段と載せたトッピングの値段の合計である。すなわち、トッピングを k 種類 $(0 \leq k \leq N)$ 載せたピザの値段は $A + k \times B$ ドルである。ピザ全体のカロリーは、生地のカロリーと載せたトッピングのカロリーの合計である。生地の値段とトッピングの値段、および、生地と各トッピングのカロリーの値が与えられたとき、「最高のピザ」の 1 ドルあたりのカロリー数を求めるプログラムを作成せよ。』

入力

『入力は $N + 3$ 行からなる。1 行目にはトッピングの種類数を表す 1 つの整数 $N(1 \leq N \leq 100)$ が書かれている。2 行目には 2 つの整数 $A, B(1 \leq A \leq 1000, 1 \leq B \leq 1000)$ が空白を区切りとして書かれている。A は生地の値段、B はトッピングの値段を表す。3 行目には、生地のカロリー数を表す 1 つの整数 $C(1 \leq C \leq 10000)$ が書かれている。$3 + i$ 行目 $(1 \leq i \leq N)$ には、i 番目のトッピングのカロリー数を表す 1 つの整数 $D_i(1 \leq D_i \leq 10000)$ が書かれている。』

入力例	出力例
3	37
12 2	
200	
50	
300	
100	

アルゴリズム

まずトッピングをカロリー順で降順にソートして、1 ドルあたりのカロリー数が上がり続ける限り足し続けていけば答えが出せそうです。いわゆる貪欲法ですね。

普通のコード

```
n,b,c,d,p[999],e;
compare(int*x,int*y){e=*x-*y;}
main(a){
  for(scanf("%d%d%d%d",&n,&a,&b,&c);d<n;d++)scanf("%d
      ",p+d);
  qsort(p,d,sizeof(int),compare);
  for(;c/a<=p[d-1]/b;d--){
    c+=p[d-1];
    a+=b;
  }
  b=!printf("%d",c/a);
}
```

解説

先述の通り標準ライブラリはあらかじめ include されますので qsort が使えます。そのかわり qsort は第 4 引数に比較関数が必要です。それを宣言しなければならないのですがそれが上のコードの compare です。

compare は昇順ソートの場合、2 つのポインタを引数に取りそのポインタに格納されてる値を比較し、第一引数のほうが大きいなら正の数、同じなら 0、小さいなら負の数を返すものです。なので普通に第一引数から第二引数を引いた値でいいのですが、メモリ溢れなどの脆弱性を考える必要がある場合は使えません。十分気をつけましょう。今回は比較するものが共に正なのでメモリ溢れは起こりませんので使用可能です。

アルゴリズムでは降順ソートをすると言いましたが、6 行目の for 文を見ると配列 p を p[d] から p[0] へ、添字の d を減少しながら参照しています。つまり d は入力の際 n までインクリメントし続けるのでこんどは 0 までデクリメ

ントすることで昇順ソートした配列を後ろからみるという形であたかも昇順ソートのように扱っています。

この問題で使える大きなテクニックは 2 つです。一つ目は関数の**引数省略**です。

GCC は int 型なら関数の型を明示的に宣言しなくても勝手に int 型と捉えてくれます。関数の引数も同じなのですが、そのとき曖昧な宣言をした関数を使用するとき引数の数があっているかどうかというチェックをしてくれません。また、引数というのは連続してスタックに積まれます。CPU によりますが第二引数は第一引数のポインタの一つ次（+1）のポインタに格納されます。これらのことを利用して関数の引数を省略することができます。

```
compare(int*x){e=*x-**((&x)+1);}
```

*((&x)+1); が普通の場合の int *y にあたる値です。よって*y は**((&x)+1) になります。

これで安心してはなりません、もっと短くなります。C 言語において配列のある要素を参照するとき a[x] という書き方をしますがこれは*(a+x) と同値です。つまり、

```
**((&x)+1)  →  *(*(1+(&x)))  →  *(1[&x])  →  *1[&x]
```

このような変態的な変換ができます。これによって驚異的に短い compare が実現できます。

```
compare(int*x){e=*x-*1[&x];};
```

もう一つのテクニックは **EOF** による終了条件です。競技プログラミングなどの入力データはこれから何行の入力があることを示す入力をすることが多くあります。今回で言うと N ですね。これは入力数が不定なのでこれがないとどこまで読めばわからなくなりコードが組めないからですが、もし今回のように N が一個だけの場合は N を読み飛ばして EOF まで読み続けるという形を取ったほうが短くなる場合があります。

```
for(scanf("%*d%d%d%d",&a,&b,&c);scanf("%d",&p[d])!=
    EOF;d++);
```

これで短くなりました。といいたいところですがじつは EOF というのは stdio.h の中で定義されている定数です。じつは include を省略した場合、関数はきっちり include してくれるのですがマクロや定数は include しません。なので EOF は使えません。しかしこれは大した問題ではありません。定数である以上何か元の値があるわけでそれにすればいいわけです。EOF は stdio.h 内で-1 として define されています。よって

```
for(scanf("%*d%d%d%d",&a,&b,&c);scanf("%d",&p[d
    ])!=-1;d++);
```

これできちんと動きます。しかし最短ではありません。ここで出てくる重要なテクニックが**ビット演算子**です。特に **NOT 演算子~** はすごく有用な演算子です。NOT 演算子はその数字の各ケタのビットを反転させるものです。また、ここで重要になってくるのが int 型における符号の扱いです。int 型は 32bit の型ですが一番上位の桁が符号を表します。最上位が 1 なら負で 0 なら正です。また 0 は正としてあつかわれます。結果的に数値は 2^{32} を足して 2^{32} での剰余になります。つまり-1 は 11111....111 で表されます。とどのつまり何が言いたいかというと、~-1 が 0 であるということです。C 言語において 0 は false ですから

```
for(scanf("%*d%d%d%d",&a,&b,&c);~scanf("%d",&p[d]);d
    ++);
```

このように縮めることができます。そのほか&p[d] というのは d+p と同値ですので縮められます。また d と d++ を一緒にできるので

```
for(scanf("%*d%d%d%d",&a,&b,&c);~scanf("%d",p+d++));
```

最終的にここまで縮みます。

短いコード

```
b,c,d,p[999],e;
m(int*x){e=*x-*1[&x];}
main(a){
  for(scanf("%*d%d%d%d",&a,&b,&c);~scanf("%d",p+d
      ++););
  for(qsort(p,d,4,m);c/a<=p[--d]/b;a+=b)
    c+=p[d];
  b=!printf("%d",c/a);
}
```

結果

165byte

四問目

問題文

『あなたはパスタが大好物であり、毎日、晩御飯にパスタを作って食べている。あなたはトマトソース、クリームソース、バジルソースの 3 種類のパスタを作ることができる。N 日間の晩御飯の予定を考えることにした。それぞれの日に 3 種類のパスタから 1 種類を選ぶ。ただし、同じパスタが続くと飽きてしまうので、3 日以上連続して同じパスタを選んではいけない。また、N 日のうちの K 日分のパスタはすでに決めてある。入力として N の値と、K 日分のパスタの情報が与えられたとき、条件をみたす予定が何通りあるかを 10000 で割った余りを求めるプログラムを作成せよ。』

入力

『入力は $K+1$ 行からなる。1 行目には 2 つの整数 $N, K(3 \leq N \leq 100, 1 \leq K \leq N)$ が空白を区切りとして書かれている。$1+i$ 行目 $(1 \leq i \leq K)$ には 2 つの整数 $A_i, B_i(1 \leq A_i \leq N, 1 \leq B_i \leq 3)$ が空白を区切りとして書かれている。これは、A_i 日目のパスタはすでに決まっており、$B_i = 1$ のときはトマトソースであり、$B_i = 2$ のときはクリームソースであり、$B_i = 3$ のときはバジルソースであることを表す。$A_i(1 \leq i \leq K)$ は全て異なる。与えられる入力データにおいて、条件をみたす予定は 1 通り以上あることが保証されている。』

入力例	出力例
20 5	2640
10 2	
4 3	
12 1	
13 2	
9 1	

アルゴリズム

普通に全探索すれば解けますが時間がかかりすぎてしまいます。3 から N において、i 日目と i-1 日目と i-2 日目の情報さえわかっていれば判断が可能ですのでいわゆる DP、動的計画法になります。$Table[i]$[その日のパスタ][1 日前][2 日前] でその状態が何通りあるのかをあらわすとして

$$Table[i][a][b][c] = \begin{cases} 0 & (a = b = c) \\ Table[i+1][Day[i+1]][a][b] & (a \neq \text{‘?‘}) \\ Table[i][l][b][c] + Table[2][b][c] + Table[3][b][c] & otherwise \end{cases}$$

※ $Day[n]$ は n 日目のパスタの種類とする

となるような漸化式を立てれば $Table$[2][一日目][0][0] が答えになります。番兵付きの DP です。

普通のコード

```
n,k,i,j,l,days[103],table[103][4][4][4];
main(){
for(scanf("%d%d",&n,&k);i<k;i++){
  scanf("%d",&j,&l);
  days[j+1]=l;
}
  for(i=n+2;i>=2;i--){
for(j=3;j>=0;j--)
  for(k=3;k>=0;k++)
  for(l=3;l>=0;l++){
    if(i==n+2)table[i][j][k][l]=1;
    else  if(j){
        if(j==k&&k==l)table[i][j][k][l]=0;
        else  table[i][j][k][l]=table[i+1][days[i+1]][
            j][k];
        table[i][0][k][l]+=table[i][j][k][l];
    }
    table[i][j][k][l]%=10000;
}
}
n=!printf("%d\n",table[2][days[2]][0][0]);
}
```

解説

　この問題は縮めたあとと縮める前がやってることは同じでも相当見た目が変わりますのでこの状態から始めさせていただきます。table が DP のデータを

保存する配列で、days が各日のパスタの種類です。0 ならば未定です。　ここではまだたいしたテクニックは使っていません。

　まず配列の添字を短縮したいと思います。

工夫したコード

```
p,n,k,i,j,l,days[103],table[103][4][4][4];
main(){
for(scanf("%d%d",&n,&k);k--;days[j+1]=l)scanf("%d%d"
    ,&j,&l);
for(i=n+2;i>=2;i--){
    for(p=63;p>=0;p--){
      j=p/16;
      k=p/4&3;
      l=p&3;
      if(i==n+2)table[i][j][k][l]=1;
      else  if(j){
        if(j==k&&k==l)table[i][j][k][l]=0;
        else  table[i][j][k][l]=table[i+1][days[i+1]][
            j][k];
        table[i][0][k][l]+=table[i][j][k][l];
      }
    table[i][j][k][l]%=10000;
}
}
n=!printf("%d\n",table[2][days[2]][0][0]);
}
```

解説

たいして短くなっていませんがｐという変数により for 文がひとつにまとめられています。勘の良い人ならばわかるかもしれませんがこれを利用して table を [103][64] の二次配列にします。

つまり**添字をまとめる**のです。これによりあらゆる代入がひとつの式で添字を制御できます。

短いコード

```
p,n,k,i,j,l,days[103],table[103][64];
main(){
for(scanf("%d%*d",&n);~scanf("%d%d",&p,&i);days[p
    +1]=i);
for(i=n+2;i>=2;i--){
  for(p=63;p>=0;p--){
    if(i==n+2)table[i][p]=1;
    else if(p/16){
      if(p%21==0)table[i][p]=0;
      else table[i][p]=table[i+1][days[i+1]*16+p/4];
      table[i][p%16]+=table[i][p];
      }
    table[i][p]%=10000;
    }
}
n=!printf("%d\n",table[2][days[2]*16]);
}
```

解説

ずいぶんとすっきりしました。添字をひとつにまとめても、ずらすなどの行動は簡単な演算で実行できます。例えば元のコードで [j][k][l] から [0][k][l] を参照したいとき

j*16+k*4+l

を

k*4+l

にするので 16 での剰余を求めればいいですね。

ほかにも

if(j==k&&k==l)

の部分ですが、これを二次元配列の方の数字と対応させると

j*16+k*4+l

になります。ここで

j==k==l

であることを利用すると、

j*16+k*4+l = j*16+j*4+j = j*21

よってこれに当てはまるものは 21 で割り切れることになります。それが

p%21==0

の部分です。

このように添え字をひとつにまとめることで、かえって処理が簡単になることがあります。

それでは細部を縮めて行きましょう。

短いコード

```
p,n,i,days[103],table[103][64];
main(){
for(scanf("%d%*d",&n);~scanf("%d%d",&p,&i);days[p+1]=
    i);
```

```
for(i=n+3;--i;){
for(p=64;p--;){
    if(i==n+2)table[p][i]=1;
    else if(p/16){
      table[p%16][i]+=table[p][i]=p%21?table[days[i
          +1]*16+p/4][i+1]:0;
    }
  table[p][i]%=10000;
  }
}
n=!printf("%d\n",table[2][days[2]*16]);
}
```

解説

特に変わったのは 8 行目の部分です。条件分岐を三項演算子にして一つの式にしました。また EOF の判断による読み込み終了をすることによって入力の行数の情報は必要無いので読み飛ばしました。

さらに縮めます。

3.0.1 短いコード

```
p,n,i,t[999][999];
main(){
for(scanf("%d%*d",&n);~scanf("%d%d",&p,&i);*t[p+1]=i
    );
for(i=n+3;--i;)
for(p=64;p--;t[p][i]%=10000)
i-n-2?p/16?t[p%16][i]+=t[p][i]=p%21?t[*t[i+1]*16+p
    /4][i+1]:0:0:t[p][i]++;
```

```
n=!printf("%d\n",t[*t[2]*16][2]);
}
```

解説

この DP のなかで table[x][0] は参照されていません。なので table[x][0] を days の代わりに使うことで変数を減らしました。また table[x][0] は*table[x] という形で短くかけるので、それも利用しました。また三項演算子をつかって if 文をなくしました。これによって式がひとつになり for 文の中括弧がいらなくなります。ついでに変数名をすべて一文字にしました。

最後まで短くします。

最短コード

```
p,n,i,t[999][999];
main(){
for(scanf("%d%*d",&n);~scanf("%d%d",&p,&i);*t[p]=i);
for(i=n+3;--i;)for(p=64;p--;t[p][i]%=10000)
i-n-2?p/16?t[p%16][i]+=t[p][i]=p%21?t[*t[i]*16+p/4][i
    +1]:0:0:t[p][i]++;
n=!printf("%d\n",t[*t[1]*16][2]);}
```

解説

元のコードの days にあたる*t の添字に常に +1 がついていたのでそれを除きました。これがおそらく最短コードです。

結果

225byte

五問目

問題文

『JOI 社の建物は図のような 1 辺 1 メートルの正六角形をつなぎ合わせた形である。クリスマスが近づいているので、JOI 社では建物の壁面をイルミネーションで飾り付けることにした。ただし、外から見えない部分にイルミネーションを施すのは無駄なので、イルミネーションは外から建物の中を通らずに行くことのできる壁面にのみ飾り付けることにした。

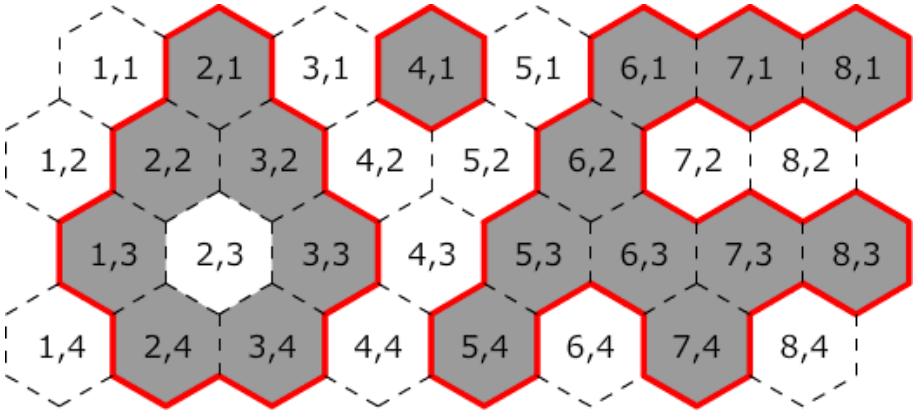

図: JOI 社の建物の配置の例

上の図は上空から見た JOI 社の建物の配置の例である。正六角形内の数字は座標を表す。灰色の正六角形は建物がある場所を表し、白色の正六角形は建物がない場所を表す。この例では、赤の実線で示される部分がイルミネーションで飾り付けを行う壁面となり、その壁面の長さの合計は 64 メートルとなる。

JOI 社の建物の配置を表す地図が与えられたとき、飾り付けを行う壁面の長さの合計を求めるプログラムを作成せよ。ただし、地図の外側は自由に行き来できるものとし、隣接した建物の間は通ることはできないものとする。』

入力

『入力ファイルの 1 行目には 2 つの整数 $W, H (1 \leq W \leq 100, 1 \leq H \leq 100)$ が空白を区切りとして書かれている。続く H 行には JOI 社の建物の配置が書かれている。$i + 1$ 行目 $(1 \leq i \leq H)$ には W 個の整数が空白を区切りとして書かれており、j 個目 $(1 \leq j \leq W)$ の整数は座標 (j, i) の正六角形に建物がある時は 1 であり、ない時は 0 である。また、与えられる入力データには建物が必ず 1 つ以上ある。

地図は以下の規則によって記述されている。

- 地図の最も北の行の最も西の正六角形は座標 $(1, 1)$ である。
- 座標 (x, y) の正六角形に隣接する東隣の正六角形は座標 $(x + 1, y)$ である。
- y が奇数の時、座標 (x, y) の正六角形に隣接する南西の正六角形の座標は $(x, y + 1)$ である。
- y が偶数の時、座標 (x, y) の正六角形に隣接する南東の正六角形の座標は $(x, y + 1)$ である。

』

入力例	出力例
8 5 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 1 0 0 1 1 1 1 1 0 1 0 1 1 0 1 0 0 1 1 0 1 1 0 0	56

アルゴリズム

範囲が大きくても 10^4 しかないということから再帰で全部探索すればいいでしょう。空いてる場所からまわりの 6 方向のうちまだ探索していない方へ進んでいき、壁があったら 1 足す、ということをすればいいでしょう。周りをつながっているようにするには番兵を回りにしけばよいでしょう。

壁なら 1、未調査なら 0、調査済みなら 2 ということで再帰関数を作ります。

普通のコード

```
l[999][999],i,j,h;
s(a,b){
  if(a<0 || b<0 || a>i+1 || b>j+1 || l[a][b]==2)
      return 0;
  else if(l[a][b]==1)return  1;
  else{
    l[a][b]=2;
  if(b%2==0){
    return  s(a-1,b-1)+s(a,b-1)+s(a-1,b)+s(a+1,b)+s(a
        -1,b+1)+s(a,b+1);
    }
    else{
    return  s(a,b-1)+s(a+1,b-1)+s(a-1,b)+s(a+1,b)+s(a
        ,b+1)+s(a+1,b+1);
    }
  }
}
main(w){
```

```
    for(scanf("%d%d",&w,&h);j++-h;)
        for(i=0;i++-w;scanf("%d",&l[i][j]));
    printf("%d",s(0,0));
}
```

解説

このアルゴリズムはひとつの関数で 6 つ再帰していることと、分岐の条件が複雑であることが難点です。しかし凄くコンパクトな形に収まっているので関数 s を縮めていく方針でやっていきたいと思います。

まず範囲外に出たかどうかの判定ですが、引数は 1 づつしか増減しないので必ず a==-1,b==-1,a==i+2,b==j+2 のどれかで弾かれることになります。なので分岐の条件をこれらに入れ替えても問題ありません。

短いコード

```
s(a,b){
if(!~a||!~b||!(a-i-2) ||!(b-j-2)||!(l[a][b]-2))return
    0;
else if(l[a][b]==1)return 1;
else{
    l[a][b]=2;
    if(b%2==0){
    return  s(a-1,b-1)+s(a,b-1)+s(a-1,b)+s(a+1,b)+s(a
        -1,b+1)+s(a,b+1);
    }
    else{
    return  s(a,b-1)+s(a+1,b-1)+s(a-1,b)+s(a+1,b)+s(a
        ,b+1)+s(a+1,b+1);
    }
```

```
    }
}
```

解説

中の条件がすべて否定で、or でつながっていますのでド・モルガンの定理よりこれを else に持って行ったら楽になることがわかります。

条件分岐の部分を変えます。また b%2 による条件分岐が冗長なので一つの式にまとめます。

短いコード

```
l[999][999],i,j,h;
s(a,b){
if(~a&&~b&&(a-i-2)&&(b-j-2)&&(l[a][b]-2)){
if(l[a][b]==1)return 1;
else
return l[a][b]=2,s(a-(b-1)%2,b-1)+s(a+1-(b-1)%2,b-1)+
    s(a-1,b)+
s(a+1,b)+s(a-(b+1)%2,b+1)+s(a+1-(b+1)%2,b+1);
}
else return 0;
}
```

解説

条件分岐は大していじくっていませんが、b%2 の扱いが少々面倒になっています。元のコードと見比べると b%2 における挙動が一致していると思います。暇な方は確認してみてください。

最後に b が b-1 から b+1 に変化していることを利用して b を動的に変えることにより短縮させます。

最短のコード

```
l[999][999],i,j,h;
s(a,b){
h=l[a][b];
h=~a*~b*(a-i-2)*(b-j-2)*(h-2)?
h-1?
l[a][b--]=2,s(a-b%2,b)+s(a+1-b%2,b)+s(a-1,++b)+
s(a+1,b)+s(a-b%2,++b)+s(a+1-b%2,b)
:1
      :0;
}
main(w){
   for(scanf("%d%d",&w,&h);j++-h;)
      for(i=0;i++-w;scanf("%d",&l[i][j]));
   printf("%d",s(0.));
}
```

解説

&&演算子をすべて乗算に置き換えました。また変数を返す部分に代入のテクニックを使っています。そして条件を三項演算子に適応するために少し形を変えました。

そして重要なテクニックとして**引数のオーバーフロー**を使用しています。最後の

printf(“%d”,s(0.));

の部分ですが、s の宣言が型を明示的に指定していないのでコンパイル時にエラーは吐かれません。また 0. は double 型の 0 を表しますのでスタックには第一引数の場所に int の二倍の長さの bit がすべて 0 になって入ります。する

と、第一引数と第二引数はポインタが隣接しているので第二引数にも 0 が入ります。よって

s(0,0)

と同じことが起こります。これによって 1byte 短縮できます。重要なテクニックです。

結果

251byte

4　番外編 JOI 本選

JOI 予選ついでに本選も縮めましょう。本選は予選ほど簡単な問題ばかりではないので 5 問中 3 問だけ shortcoding します。

第一問

問題文

『JOI (日本情報オリンピック) の本選に向けてプログラミングの練習をしていたあなたは、今年度の JOI の予選の問題には数値を扱う問題ばかりが出題され、文字列を扱う問題がなかったことに気がついた。そこであなたは、こっそり文字列の問題に強くなってライバルたちに差をつけることにした。

JOI の過去問を眺めていると、J、O、I の 3 種類の文字からなる文字列に慣れておく必要がありそうなことがわかった。そこで、そのような文字列について考えよう。あなたは「与えられた文字列が JOI という部分文字列をもつかどうかを答えよ」という問題を思いついたものの、これはすぐに解けてしまった。もっとレベルの高い問題を解きたいあなたは、以下のような問題を作った。

文字列 *t* が文字列 *s* の部分文字列であるとは、*t* の先頭および末尾に何文字か (0 文字でもよい) を付け足すと s になることである。たとえば、JJOOII は OJJOOIIOJOI の部分文字列である。一方、JOI は JOOI の部分文字列では

ない。

また、0 以上の整数 k に対し、レベル k の JOI 列とは、k 個の文字 J、k 個の文字 O、k 個の文字 I をこの順に並べた文字列のことであるとする。たとえば、JJOOII はレベル 2 の JOI 列である。

与えられた文字列の部分文字列である JOI 列のうち、レベルが最大のものを求めたい。』

入力

1 行目に J、O、I の 3 種類の文字からなる文字列 S が書かれている。

制限

$1 \leq N \leq 1000000 (= 10^6)$　　S の長さ

入力例	出力例
JJOOII	2

アルゴリズム

左から順番に連続して並ぶ J の数、O の数、I の数を順番にとっていき、O が最小のとき、O の個数がレベルになりますので、それの最大値を更新しながら計算していきます。オーダーは N です。

普通のコード

```
j,o,i,n;
main(c){
    for(c=getchar();c-10;){
      for(j=o=i=0;c=='J';c=getchar())j++;
      for(;c=='O';c=getchar())o++;
      for(;c=='I';c=getchar())i++;
      if(o<=fmin(j,i))n=n>o?n:o;
```

```
    }
    n=!printf("%d\n",n);
}
```

ちょっと短いコード

```
j,o,i,n;
main(c){
    for(c=getchar();c-10;){
      for(j=o=i=0;c-74;c=getchar())j++;
      for(;c-79;c=getchar())o++;
      for(;c-73;c=getchar())i++;
      if(o<=fmin(j,i))n=n>o?n:o;
    }
    n=!printf("%d\n",n);
}
```

解説

J、O、I に対していいかんじの規則性（剰余など）がないのでループで回せないと思うかもしれません。しかし”JOI”という char[] を用意すると getchar を 4 回も記述しなくてよくなります。

割と短いコード

```
j[3],n,t;
main(c){
for(c=getchar();c-10;t?:fmin(j[1],*j)>=j[2]?n=fmax(n,
    j[2]):1)
for(t++||memset(j,0,12);c=="IJO"[t%=3];c=getchar())j[
    t]++;
```

```
n=!printf("%d\n",n);}
```

解説

“JOI”ではなく”IJO”となるのは、t++ を単体で存在させたくないので、t が参照される他の所で実行させてしまおうと思った結果、ひとつずれたものになってしまったためです。

t%=3 と t++ が必要なので、うまい感じに配置してやりました。

このように char[] を使うことによって無秩序な文字列をループ化できます。また、getchar が 2 つもありますが、与えられる文字列 S の前に余分な’I’がつくものと考えればひとつめを c=73 に変えることができます。

最短コード

```
j[3],n,t;
main(c){
for(c=73;c-10;t?:fmin(j[1],*j)>=j[2]?n=fmax(n,j
    [2]):1)
for(t++||memset(j,0,12);c=="IJO"[t%=3];c=getchar())j[
    t]++;
n=!printf("%d\n",n);}
```

結果

151byte

Tips

今回 memset を使用していますが 0 で初期化する場合は **bzero** という関数がつかえます。ただしこれは環境依存なので使えたり使えなかったりします。memset(a,0,c) を bzero(a,c) に出来ますので 3byte の短縮になります。場合によっては使えるのでぜひ心に留めておいてください。

第二問

問題文

『1 から 1000 までのどれかの整数が書かれたカードがたくさんある。アンナとブルーノはそれらのカードを用いて、次のようなゲームをする。アンナは A 枚、ブルーノは B 枚のカードからなる山を持つ。アンナは A 枚のカードの中から任意の何枚か（0 枚でもよい）を捨てて新しい山を作る。ブルーノは B 枚のカードからなる山の一番上から何枚か（0 枚でもよい）と、一番下から何枚か（0 枚でもよい）を捨てて新しい山を作る。ただし、捨てる際に残ったカードの並び替えは行わない。このように作った 2 つの山が一致していたら、一方の山に含まれるカードの枚数が 2 人の得点になる。ただし、2 つの山が一致するとは、山に含まれるカードの枚数 n が同じで、かつ上から i 番目 $(1 \leq i \leq n)$ に書かれたカードの整数が全て同じであることである。例えば、アンナが 5 枚のカードの山を持ち、書かれている整数は上から順に 1、2、3、4、5 であり、ブルーノが 4 枚のカードの山を持ち、書かれている整数が上から順に 3、1、4、1 であったとする。このとき、アンナが 2、3、5 のカードを捨て、ブルーノが一番上の 3 と一番下の 1 のカードを捨てると 2 人の山が一致する。このとき、残った山の一方に含まれるカードの枚数は 2 枚なので、2 人は得点 2 を得る。2 人の得点の最大値を求めたい。アンナとブルーノが持っているカードの山の情報が与えられたときに、2 人の得点の最大値を求めるプログラムを作成せよ。』

入力

1 行目には、整数 A, B が空白を区切りとして書かれている。

2 行目には、A 個の整数が空白を区切りとして書かれており、i 番目の整数 $(1 \leq i \leq A)$ はアンナの持っている山の上から i 番目のカードに書かれている整数を表す。

3 行目には、B 個の整数が空白を区切りとして書かれており、j 番目の整数

$(1 \leq j \leq B)$ はブルーノの持っている山の上から j 番目のカードに書かれている整数を表す。

制限

$1 \leq A \leq 5000$

$1 \leq B \leq 5000$

カードに書かれている整数は 1 以上 1000 以下である.

アルゴリズム

アンナではなくブルーノの山を基準に考えると、ブルーノが i 番目からとったときに取りうる枚数の最大値を求める作業を各 i で行えば、オーダーは $A*B$ となり、今回は値が小さいのでこれで解くことができます。

入力例	出力例
5 4	2
1 2 3 4 5	
3 1 4 1	

普通のコード

```
card[2][5000],a,b,i,j,n,t;
main(){
  scanf("%d%d",&a,&b);
  for(;i-a;scanf("%d",card[0]+i++));
  for(;j-b;scanf("%d",card[1]+j++));
  for(;--j;n=n>t?n:t){
    for(i=t=0;j+t<b&&i<a;)card[0][i++]-card[1][j+t]?:
      t++;
  }
  n!=printf("%d\n",n);
```

```
}
```

ちょっと短いコード

```
card[2][5001],a,i,j,n,t;
main(){
  scanf("%d%*d",&a);
  for(;i-a;scanf("%d",card[0]+i++));
  for(;~scanf("%d",card[1]+j++););
  for(;--j;n=n>t?n:t){
    for(i=t=0;i<a;)card[0][i++]-card[1][j+t]?:t++;
  }
  n!=printf("%d\n",n);
}
```

解説

ちょっと短いコードは b の入力の終了条件が EOF によるもので充分だということを利用しています。すると、変数 b がいらないことがわかりました。

また探索部（7 行目）の for 文の終了条件ですが、card 配列を余分に取ることで番兵をつくり、終了条件を短くしました。

まだ冗長なので配列を短くします。

最短のコード

```
card[2][9999],a,i,j,n,t;
main(){
  scanf("%d%*d",&a);
  for(;i-a;scanf("%d",*card+i++));
  for(;~scanf("%d",card[1]+j++););
  for(;--j;n=n>t?n:t)
    for(i=t=0;i<a;)i++[*card]-card[1][j+t]?:t++;
```

```
    n!=printf("%d\n",n);
}
```

解説

ここでも shortcoding で重要な a[x]=*(a+x) を利用したテクニックが使われています。card[0][i++] ですがポインタの間接演算子*よりも添字参照 [] のほうが優先度が高いので*card[i++] とはできません。なので (*card)[i++] としなければならないのですが、括弧が冗長です。カッコを使わないでいいように i++[*card] としています。

結果

197byte

第四問

問題文

http://www.ioi-jp.org/joi/2011/2012-ho-prob_and_sol/2012-ho.pdf

アルゴリズム

釘 A, B にたいして大きさ C の三角形がかかっていた場合、釘 $A+1, B$ と釘 $A+1, B+1$ に大きさ $C-1$ かかっていると考えても大丈夫なので、各釘にかかっている三角形を DP することで、オーダー N^2 になります。具体的には、ある頂点 A, B に対してかかっている三角形の大きさを $S_{a,b}$ とすると、

$$S_{a,b} = max(S_{a-1,b} - 1, S_{a-1,b-1} - 1, S_{a,b})$$

という関係式が成り立ちます。これで DP します。

普通のコード

```
kugi[5001][5001],a,b,res,x,y,z,i,j;
```

```
main(){
  scanf("%d%d",&a,&b);
  for(;~scanf("%d%d%d",&x,&y,&z);){
    kugi[x][y]=z+1;
  }
  for(;i++<a;){
    for(j=0;j++<a;){
      kugi[i][j]=fmax(kugi[i][j],fmax(kugi[i-1][j],
          kugi[i-1][j-1])-1);
      if(kugi[i][j])res++;
    }
  }
  a=!printf("%d\n",res);
}
```

ちょっと短いコード

```
k[5001][5001],a,b,r,x,y,z,i,j;
main(){
  for(scanf("%d%*d",&a);~scanf("%d%d%d",&x,&y,&j);){
    kugi[x][y]=j+1;
  }
  for(;i++<a;){
    for(j=0;j++<a;){
      k[i][j]=fmax(k[i][j],fmax(k[i-1][j],k[i-1][j
          -1])-1);
      r+=!!k[i][j];
    }
```

```
    }
    a=!printf("%d\n",r);
}
```

解説

今回のミソであるテクニックは**二重否定**です。これは!0==1 ということを利用したもので a?1:0 というのを!!a で代用できるというものです。!!によって値が 0 であるものは 0、それ以外は 1 になります。r+=!!k[i][j] の部分はそれを利用しています。

わりと短いコード

```
k[5001][5001],a,b,r,x,y,z,i,j;
main(){
    for(scanf("%d%*d",&a);~scanf("%d%d%d",&x,&y,&j);)k[
        x][y]=j+1;
    for(;i++<a;)for(j=0;j++<a;)
r+=!!(k[i][j]=fmax(k[i][j],fmax(k[i-1][j],k[i-1][j
      -1])-1));
    a=!printf("%d\n",r);
}
```

解説

代入演算子の戻り値は代入後の値であることを利用してこのように縮めることができます。

次は二重 for 文が冗長なので縮めましょう。二重 for 文を縮めるには

for(a;b;c)for(d;e;f)g;

が

for(a;e?g,f:(c,d,b););

とほぼ同値であることを利用します。a から g 全てに何か式が入っていたら

縮まりませんが、b,c,d,f,g のうちどれかがない場合コンマがその文だけ消えるので、これによって短縮できます。この置換は (g,f) が常に真であることと、b と d の実行順序が逆転しても問題ないときに、利用可能です。今回は f が無いので g が常に真でないといけないのですが、r は 0 で初期化され、r+=0 されたら返り値が 0 なので必ずしも真ではありません。なので 1 で初期化して出力時に 1 引くようにします。

わりと短いコード

```
k[5001][5001],a,b,x,y,z,i=1,j;
main(r){
for(scanf("%d%*d",&a);~scanf("%d%d%d",&x,&y,&j);)k[x
    ][y]=j+1;
for(;j++<a?r+=!!(k[i][j]=fmax(k[i][j],fmax(k[i-1][j],
    k[i-1][j-1])-1)):(j=0,i++<a););
a=!printf("%d\n",r-1);
}
```

解説

i の初期値が 1 に変わっていますがこれは i++ の実行順序が k[i][j] 参照時より後になったからです。これをしないと k[-1] を参照する事態が発生し、予期せぬ動作になっていまいます。

最後にこのアルゴリズムでは a の値は実は必要ありません。なぜなら a が大きい分には釘の数が増えるだけで三角には影響が無いので問題ないからです。なので scanf がひとつ消せます。

最短のコード

```
k[5001][5001],x,y,z,i=1,j;
main(r){
```

```
for(gets();~scanf("%d%d%d",&x,&y,&j);)k[x][y]=j+1;
for(;j++<5e2?r+=!!(k[i][j]=fmax(k[i][j],fmax(k[i-1][j
    ],k[i-1][j-1])-1)):(j=0,i++<a););
x=!printf("%d\n",r-1);
}
```

結果

193byte

5　shortcode を通して

shortcode は脆弱性だけを生むものではないです。短くなったコードは読みにくく、コンパイラの負担も大きかったりして、特に利点はないように見えます。たしかに娯楽性だけがずば抜けている行為だとは思いますが、shortcoding をすることで言語仕様に詳しくなったり、アルゴリズムに強くなったりします。コードが短ければ単純なアルゴリズムでメモリ使用量も少ないでしょう。たぶん。なので一概に実用性のないものとは言えません。いや実用性はありません。だけど有益です。みなさんもアルゴリズムに強くなりたかったり言語仕様を極めたい時はぜひ、shortcoding をしてみてください。

また shortcoding は C 言語に限ったものではありません。Ruby や Python などいろいろな言語で shortcoding は密かに流行っています。もし本記事で C の shortcoding に向いてないと思っても他の言語では十分 shortcoding を楽しめるかもしれません。ぜひやってみてください。

6　追記

この記事のショートコードはほとんど数時間で組んだものなので最短である保証はありません。私のコードより短いものができた場合は byte 数だけでもご報告ください。Twitter(@catupper) でも Facebook(小倉拳) でも。