三角の綻び
解　説

添付ファイルをダウンロードすると

zipファイルが確認できます。以下の

コマンドでzipファイルを解凍すると

challenge.c とTriangle.png が確認

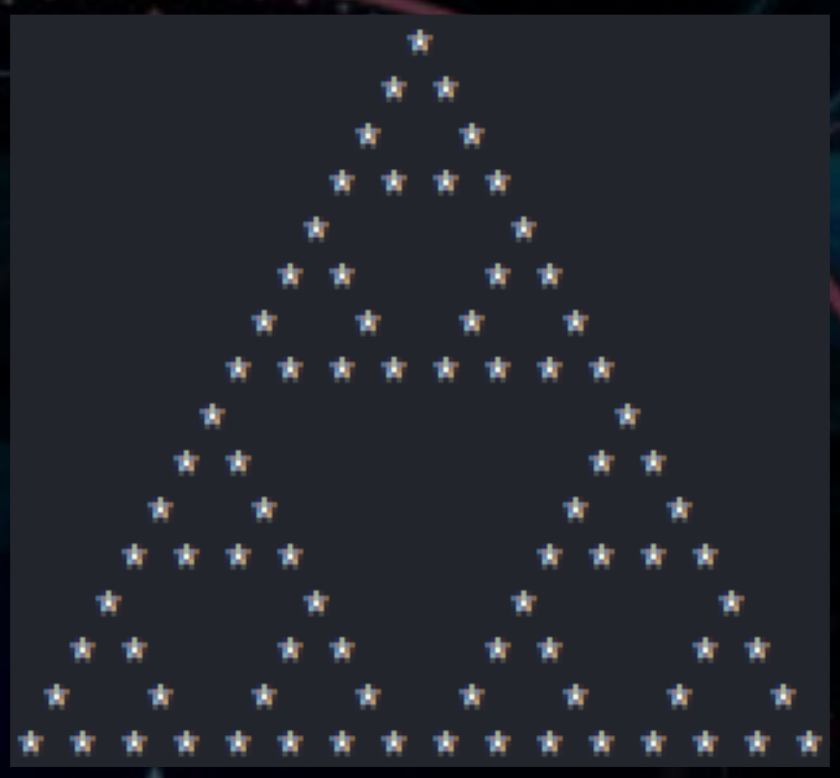
できます。

```
┌──(kali㉿kali)-[~/triangle]
└─$ unzip triangle_pattern.zip
Archive:  triangle_pattern.zip
   creating: triangle_pattern/
  inflating: triangle_pattern/Triangle.png
  inflating: triangle_pattern/challenge.c

┌──(kali㉿kali)-[~/triangle]
└─$ ls -la triangle_pattern
合計 20
drwxr-xr-x 2 kali kali 4096 12月  22 17:46 .
drwxrwxr-x 4 kali kali 4096 12月  23 11:56 ..
-rw-r--r-- 1 kali kali 8127 12月  22 16:19 Triangle.png
-rw------- 1 kali kali 2199 12月  22 17:46 challenge.c
```

配布されたファイルは challenge.c

というCのソースコードです。

問題文によると、三角形パターンを生成するプログラムですが、正しく動作していないとのことです。
Triangle.pngを確認します。



問題文に添付されている画像を確認すると、正しく生成された場合の最初の16行のパターンが示されています。
この画像のパターンを見ると、
フラクタル図形の一種である
シェルピンスキーの三角形」である
ことがわかります画像です。

次に、配布されたコードを改めて確認します。コード内のコメントを見ると、「このプログラムは三角形のパターンを生成します。しかし、正しく動作していません。デバッグして修正し、正しいパターンを生成して下さい。」との記載が確認できます。期待される出力は先程表示された画像です。

```
/*
 * Triangle Pattern Generator
 *
 * このプログラムは三角形パターンを生成します。
 * しかし、正しく動作していません。
 * デバッグして修正し、正しいパターンを生成してください。
 *
 * 期待される出力は問題文の画像を参照してください。
 */
```

実際にgccにてコンパイルして実行してみます。

```
┌──(kali㉿kali)-[~/triangle]
└─$ gcc -o challenge challenge.c

┌──(kali㉿kali)-[~/triangle]
└─$ ./challenge
```

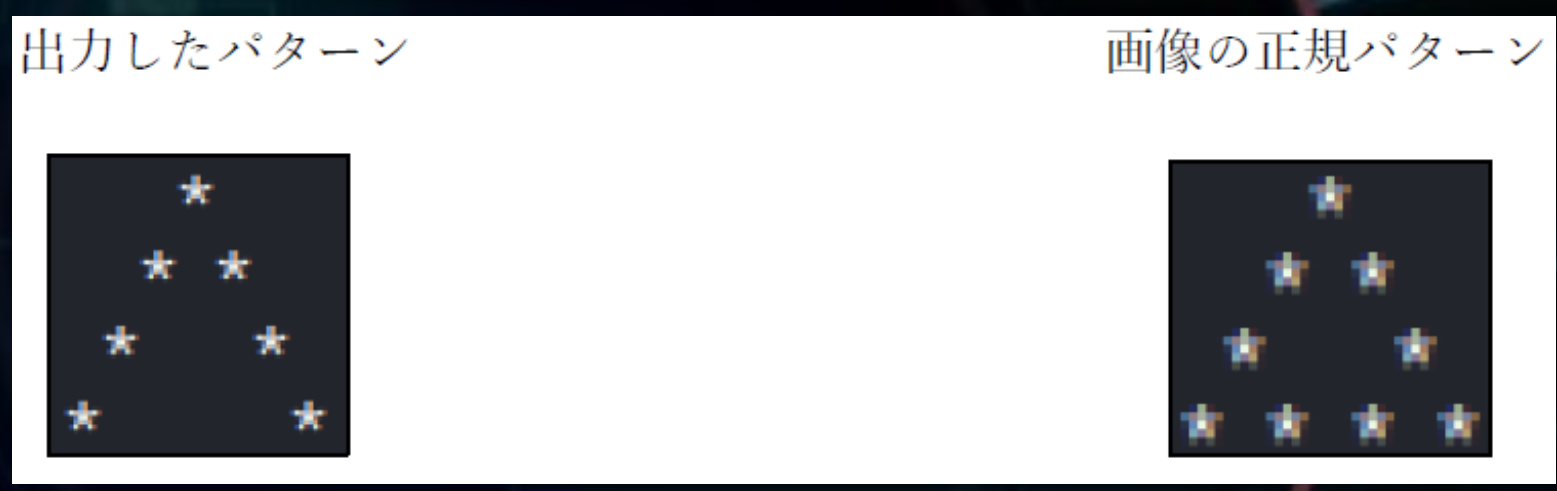
プログラムを実行すると、三角形のパターンが画面に表示されます。
64行の大きな三角形で、＊と空白で描かれています。

```
┌──(kali㉿kali)-[~/triangle/triangle_pattern]
└─$ ./challenge
═══ Triangle Pattern Challenge ═══

Triangle Pattern:
(*=1, space=0)
```

最後に以下の統計情報が表示されます。

```
Statistics:
32行目: 7 stars
64行目: 0 stars

Result: 7000
Flag: flag{7000}
```

実際の出力とコメント内の期待されるパターンを比べてみると、4行目から明らかに違います。期待される4行目は＊＊＊＊となっていますが、実際の出力では真ん中が抜けています。



他の行でもパターンが崩れていることがわかります。つまり、このプログラムには確かに不具合があり、直す必要があります。シェルピンスキーの三角形について調べてみると、パスカルの三角形を2で割った余り（mod 2）で表現されるフラクタル図形であることがわかります。

パターン生成部分を確認すると、以下のようになっています。

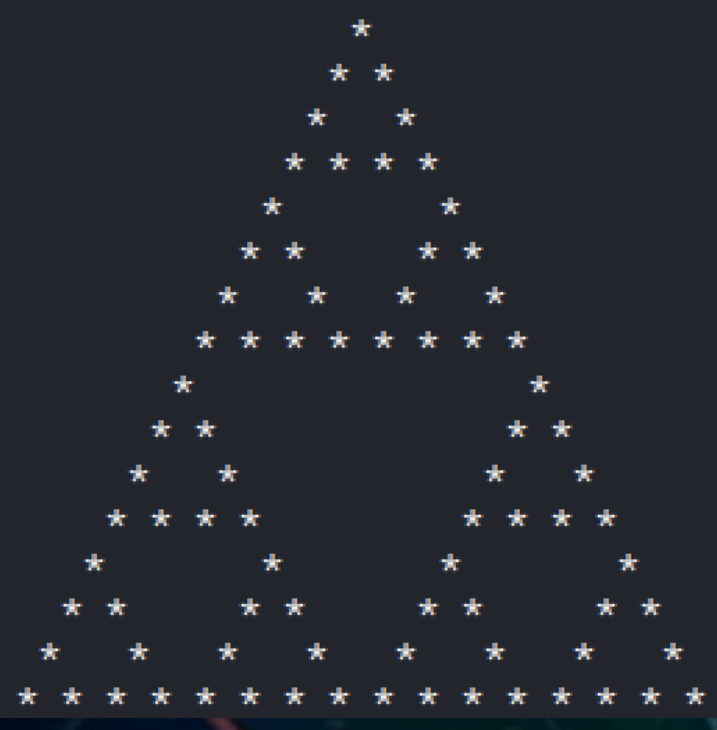
```
for(int j = 1; j < i; j++) {
    triangle[i][j] = (triangle[i-1][j-1] + triangle[i-1][j]) % 3;
}
```

つまり、各要素は0または1の2値で表現される必要があります。ところが、現在のコードでは% 3となっているため、0、1、2の3つの値が生成されてしまい、正しいパターンが生成されません。% 2に直します。

```
for(int j = 1; j < i; j++) {
    triangle[i][j] = (triangle[i-1][j-1] + triangle[i-1][j]) % 2;
}
```

直したあとに再度コンパイルして実行すると、パターンは対称的になりました。

```
┌──(kali㉿kali)-[~/triangle/triangle_pattern]
└─$ ./challenge
═══ Triangle Pattern Challenge ═══

Triangle Pattern:
(*=1, space=0)
```



しかし、統計情報を見ると以下のようになっています。

```
Statistics:
32行目: 1 stars
64行目: 0 stars

Result: 1000
Flag: flag{1000}
```

パターンは正しくなりましたが、
統計の値を確かめる必要があります。
正しく生成された場合、32行目は
1個しかカウントされていません。

これは1行目以外では考えられない数値です。64行目も同じで、最低でも複数あるはずですが、0個とカウントされています。
count_ones_in_row()関数を見ると、以下のようになっています。

```
int count_stars_in_row(int row) {
    int count = 0;
    for(int j = 0; j < row; j++) {
        if(triangle[row][j] == 1) {
            count++;
        }
    }
    return count;
}
```

仕様によると、i行目には0からi列目まで（0 <= j <= i）の要素が存在します。

しかし今のコードではj < rowと
なっているため、最後の要素
（row列目）を数え忘れているため、
64行目の個数が0個になっていると
考えられます。j <= rowに直します。

```
int count_stars_in_row(int row) {
    int count = 0;
    for(int j = 0; j <= row; j++) {
        if(triangle[row][j] == 1) {
            count++;
        }
    }
    return count;
}
```

直したあとに再度実行すると、以下
のようになります。

```
Statistics:
32行目: 2 stars
64行目: 0 stars

Result: 2000
Flag: flag{2000}
```

64行目の統計は依然として0個で、
32行目の個数が2個になりました。
32行目は合っているかもしれませんが、
64行目は間違いなく間違っています。
32行目のカウントと64行目の
カウントを取り出す際のコード内を
見る以下のようになっています。

```
// 32行目と64行目の*の数を数える
int stars_row32 = count_stars_in_row(32);
int stars_row64 = count_stars_in_row(64);
```

32行目をカウントするつもりですが、
count_ones_in_row(32)を呼んでいます。
しかし、配列は0から始まるため、
32行目は index 31 です。

count_ones_in_row(31)に直します。
同じく、64行目も
count_ones_in_row(63) に直します。

```
// 32行目と64行目の*の数を数える
int stars_row32 = count_stars_in_row(31);
int stars_row64 = count_stars_in_row(63);
```

すべての不具合を直したあと、最後に
実行すると、以下の結果が得られます。

```
Statistics:
32行目: 32 stars
64行目: 64 stars

Result: 32064
Flag: flag{32064}
```

flagが得られました。

flag{32064}