入門講習会 第三回

1. 今回の目標

   ABC081 B

   が解けるレベルの文法を学ぶ.

   問題を見てみよう.

ABC081B - Shift only

時間制限 : 2sec / メモリ制限 : 256MB

配点 : 200 点

問題文

黒板に N 個の正の整数 $A_1,\ldots,A_N$ が書かれています.

すぬけ君は, 黒板に書かれている整数がすべて偶数であるとき, 次の操作を行うことができます.

- 黒板に書かれている整数すべてを, 2 で割ったものに置き換える.

すぬけ君は最大で何回操作を行うことができるかを求めてください.

制約

- $1 \leq \mathrm{N} \leq 200$
- $1 \leq \mathrm{A_i} \leq 10^9$

入力

入力は以下の形式で標準入力から与えられる.

N

$A_1\ A_2\ \ldots\ A_N$

出力

すぬけ君は最大で何回操作を行うことができるかを出力せよ.

(https://abc081.contest.atcoder.jp/tasks/abc081_b より)

方針としては非常に単純である.黒板上の全ての数が,2 で割り切れなくなるまで割り続

ければよい.より詳しくいえば,次のような考えで解けそうである.

・$A_1, A_2, \cdots, A_N$ の数は配列で管理する.

・求めるのは操作回数なので,それを保持しておく変数を用意しておく.

①数を **A_1 から A_N に向かって順に見ていき**,もし 2 で割り切れるなら 2 で割る.

②途中,2 で割り切れない数が出てきたとき,そこで**操作をやめる**.

③A_1 から A_N まで見ることができたら,問題文に示されている操作が 1 回できたということだから,操作回数を保持している変数に 1 加える.

④**①に戻る**

配列については第一回で扱った.2 で割り切れるとか割り切れないとかの処理は if 文で実現できる.しかし,傍線太字の処理,すなわち

・A_1 から A_N に向かって順に見ていく

・途中で操作をやめる

・同じことを繰り返す

を今までの知識だけで実現するのは不可能である.そこで今回は,同じことを繰り返すための「ループ文」について学ぼう.

ループ文には while 文・do-while 文・for 文の三種類がある.順に説明しよう.

2. while 文

最も単純な while 文から説明する.以下の書式で書く.

```
while（継続条件）処理
```

継続条件が 0 でない間処理をし続ける.{}でくくると複数の処理が書ける.

例を示そう.以下は,「Hello」と 10 回出力するプログラムである.

```c
#include <stdio.h>

int main(void) {
    int i;

    i = 0;
    while (i < 10) {
        printf("Hello.¥n");
        i++;
    }

    return 0;
}
```

実行結果は以下のようになる.

```
Hello.
Hello.
Hello.
Hello.
Hello.
Hello.
Hello.
Hello.
Hello.
Hello.
```

while 文の動きを追ってみよう.

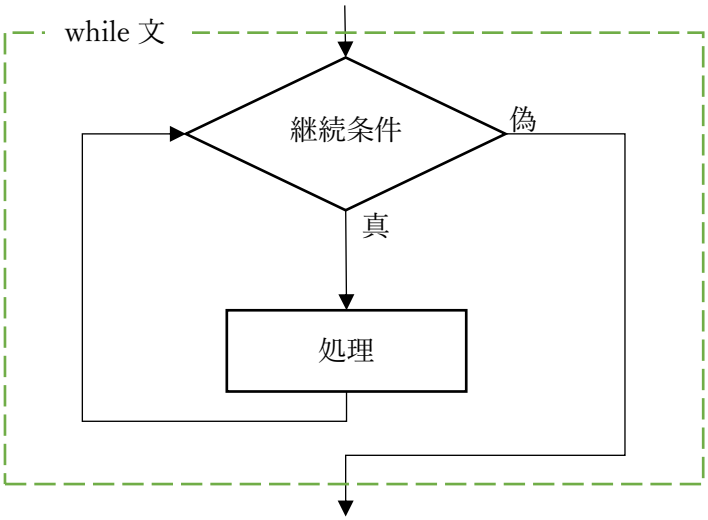
まず while の手前で i には 0 が代入されている.while の継続条件は i < 10 である.つまり i が 10 未満の間, { } 内の処理は繰り返されることになる.{ }内では,まず printf で「Hello.」と表示し,そして i をインクリメントしている.このことを考えると,次のような流れで処理が行われることが分かる.

i が 0 のとき,「Hello」と出力して i をインクリメント.i は 1 になる.
i が 1 のとき,「Hello」と出力して i をインクリメント.i は 2 になる.
i が 2 のとき,「Hello」と出力して i をインクリメント.i は 3 になる.
i が 3 のとき,「Hello」と出力して i をインクリメント.i は 4 になる.
i が 4 のとき,「Hello」と出力して i をインクリメント.i は 5 になる.
i が 5 のとき,「Hello」と出力して i をインクリメント.i は 6 になる.
i が 6 のとき,「Hello」と出力して i をインクリメント.i は 7 になる.
i が 7 のとき,「Hello」と出力して i をインクリメント.i は 8 になる.
i が 8 のとき,「Hello」と出力して i をインクリメント.i は 9 になる.
i が 9 のとき,「Hello」と出力して i をインクリメント.i は 10 になる.
i が 10 のとき,継続条件である i< 10 を満たさないので,ループを抜ける.

このように,カウント用の変数 i を用意して繰り返し文を制御することはよくあるので覚えておこう.

継続条件を見る位置は,中身の処理が行われる前である.よって上のプログラムにおいて,例えば i= 0 の部分を i = 10 に書き換えると,中身の処理は一度も実行されずプログラムが終了する.

while 文の全体を図式化すると以下のようになる.このようにプログラムの流れを表す図をフローチャートという.



3. do-while 文

```
do 処理 while (継続条件);
```

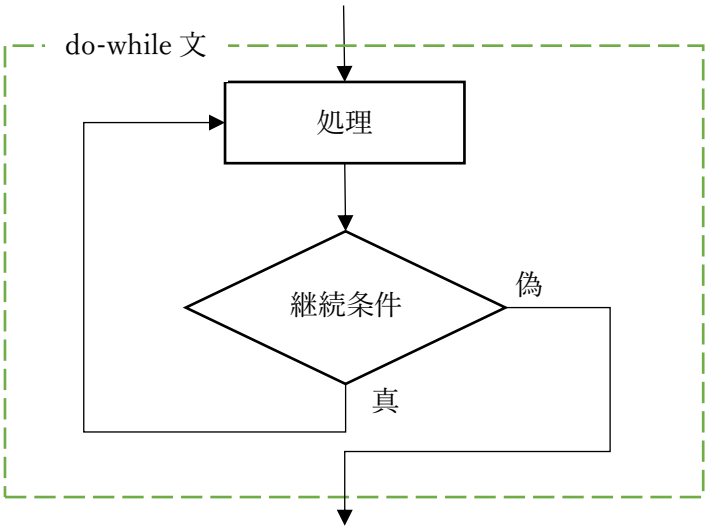
while 文とほぼ同じ.継続条件を見る位置が後ろになっているため,必ず一回は処理が実行される.最後にセミコロンをつけなくてはならない.

一応例としてソースコードを示すが,実行結果は前項のプログラムと同じである.

```c
#include <stdio.h>

int main(void) {
    int i;

    i = 0;
    do {
        printf("Hello.¥n");
        i++;
    } while (i < 10);

    return 0;
}
```

このプログラムにおいて,例えば i= 0 の部分を i = 10 に書き換えてみよう.すると,前項のプログラムとは違い,{ }の中身が一度だけ実行される.これは,i < 10 の条件を { }の後に見ているためである.

do-while 文のフローチャートは以下のようになる.

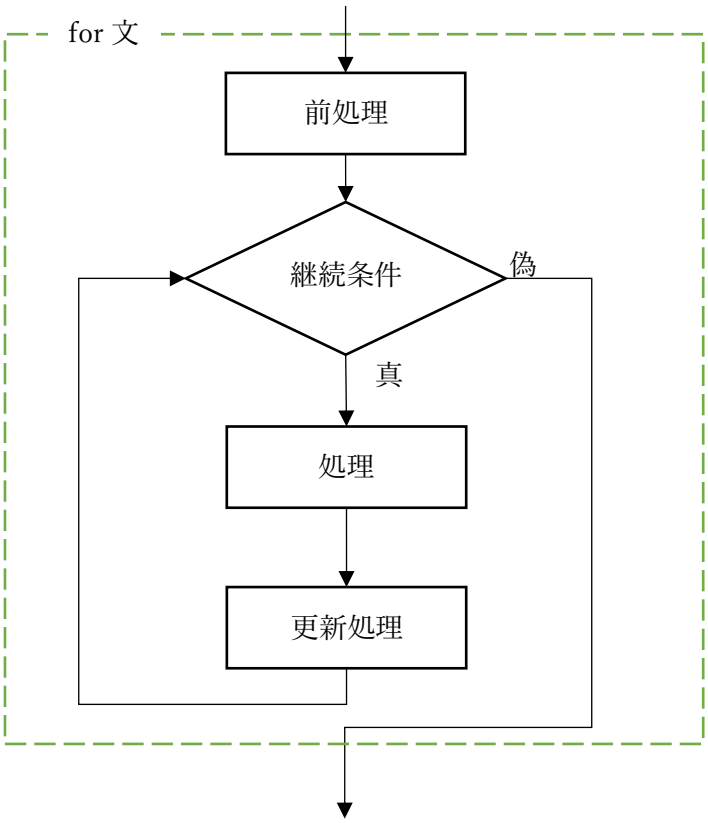


4. for 文

```
for（前処理；継続条件；更新処理）処理
```

継続条件を見る位置は,処理の前.while との違いは,前処理部と更新処理部を書く部分がある点である.前処理部ではカウント用変数の値の設定,更新処理部ではカウントを進める,という処理を書くことが多い.

フローチャートを見てみよう.前処理,継続条件,更新処理がどこで行われているかに注目して欲しい.



例を見てみよう.以下は,Hello と N 回出力するプログラムである.ついでにカウンタ用の変数の値も出力している.

```c
#include <stdio.h>

int main(void) {
    int i, N;

    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        printf("Hello%d¥n", i);
    }

    return 0;
}
```

実行結果は以下のようになる.*5* を入力例としている.

```
5
Hello0
```

```
Hello1
Hello2
Hello3
Hello4
```

変数 i をカウント用変数としている.そして,前処理で i に 0 を代入.継続条件は i<N,更新処理は i++である.

N = 5 のとき,以下の処理が行われている.

i の値に 0 をセット.
i の値が 0 のとき,「Hello, World0」と表示し,i++.
i の値が 1 のとき,「Hello, World1」と表示し,i++.
i の値が 2 のとき,「Hello, World2」と表示し,i++.
i の値が 3 のとき,「Hello, World3」と表示し,i++.
i の値が 4 のとき,「Hello, World4」と表示し,i++.
i の値が 5 のとき,継続条件である i < 5 を満たさないので,ループを抜ける.

while 文と処理が非常に似ていることが分かるだろう.一般に,while 文で書ける処理は for 文でも書けるし,その逆も成り立つ.では,どのように使い分けるべきなのだろうか.同じ処理を while 文と for 文を書いて比較してみよう.以下,変数 i, N の宣言や N の入力などは省略している.

for 文を使った場合	while 文を使った場合

`for (i = 0; i < N; i++) {`
`  printf("Hello%d¥n", i);`
`}`
	`i = 0;`
	`while (i < N) {`
	`  printf("Hello%d¥n", i);`
	`  i++;`
	`}`

i = 0 や i++の処理が,for 文では( )の中にまとまって書かれている.そのため i がカウンタ用の変数であることが明確でわかりやすい.この場合は while よりも for の書き方の方が良いと言えそうだ.

では,今後は for 文だけ使えばいいのかというとそうとも言い切れない.複雑な前処理や更新処理が行われるときは,むしろ for 文の方が分かりづらくなってしまうこともある.どんなケースで for,どんなケースで while を使うべきなのかについては,後に示す例題や演習を通して学んでほしい.

5. 多重ループ

ループ文の中にループ文を書くことがある.これは多重ループ(ループの重なりに応じ

て,二重ループ・三重ループ・…)と呼ばれる.
以下は,掛け算九九の結果を表示するプログラムである.
※printf 中に%3d という記述がある.これは最小フィールド幅を 3 に設定して桁を揃える役割を担っているのだが,詳しくは補足資料に回す.

```c
#include <stdio.h>

int main(void) {
    int i, j;

    for (i = 1; i <= 9; i++) {
        for (j = 1; j <= 9; j++) {
            printf("%3d", i * j);
        }
        printf("¥n");
    }
}
```

実行結果は以下のようになる

```
  1   2   3   4   5   6   7   8   9
  2   4   6   8  10  12  14  16  18
  3   6   9  12  15  18  21  24  27
  4   8  12  16  20  24  28  32  36
  5  10  15  20  25  30  35  40  45
  6  12  18  24  30  36  42  48  54
  7  14  21  28  35  42  49  56  63
  8  16  24  32  40  48  56  64  72
  9  18  27  36  45  54  63  72  81
```

for 文の中を見てみると,次のような処理をすることが分かる.

i = 1 のとき,j = 1, 2, … 9 について 1 * j の計算結果を表示.その後改行する.
i = 2 のとき,j = 1, 2, … 9 について 2 * j の計算結果を表示.その後改行する.
i = 3 のとき,j = 1, 2, … 9 について 3 * j の計算結果を表示.その後改行する.
i = 4 のとき,j = 1, 2, … 9 について 4 * j の計算結果を表示.その後改行する.
i = 5 のとき,j = 1, 2, … 9 について 5 * j の計算結果を表示.その後改行する.
i = 6 のとき,j = 1, 2, … 9 について 6 * j の計算結果を表示.その後改行する.
i = 7 のとき,j = 1, 2, … 9 について 7 * j の計算結果を表示.その後改行する.
i = 8 のとき,j = 1, 2, … 9 について 8 * j の計算結果を表示.その後改行する.

i = 9 のとき,j = 1, 2, … 9 について 9 * j の計算結果を表示.その後改行する.

多重ループに使うループ文の組み合わせは様々である.for の中に for を書くこともあれば,for の中に while を書いたり,while の中に for を書いたりする.使い分けについては,後に示す例題や演習を通して学んでほしい.

6. break 文

switch 文中で break を書くと,その地点で switch 文中を抜けることができることを,前回に学んだ.

ループ文中で break を用いると,その地点でループを抜けることができる.

```
break;
```

いままでは処理を抜けるために継続条件の地点まで見る必要があったのだが,break 文によって,処理の任意の位置で切りやめることができる.

例を示そう.以下は,for ループの回数は N 回のはずだが,break 文によって 7 回で以上は処理しないプログラムである.

```
#include <stdio.h>

int main(void) {
    int i, N;

    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        if (i == 7) break;
        printf("Hello%d¥n", i);
    }

    return 0;
}
```

実行結果は以下のようになる.入力例は *100* としている.

```
100
Hello0
Hello1
Hello2
Hello3
Hello4
Hello5
```

```
Hello6
```

100 を入力したので,100 回 Hello が表示されると思いきや,i == 7 のときに break 文によってループ文を抜けてしまうので,結局 7 回以上はループが実行されない.

ここで,break についての注意点を述べる.break 文は,ループ文中で用いて,そのループ一つを抜けることができるだけである.従って,例えば多重ループ内で break を一つ用いて,すべての多重ループを一気に抜けるということはできない.多重ループ全体を抜けたい場合は,適当なフラグ変数を用意しておく(例については後で述べる).

7. continue 文

continue 文は,処理をスキップするための文である.書式は以下の通りである.

```
continue;
```

これをループ中の任意の位置で用いると,以降の処理をスキップして,ループの先頭に戻ることができる.for 文の場合は,更新処理を行った後にループ先頭に戻る.
例を示そう,以下は Hello と出力するが,i が 3 の倍数の場合は処理を飛ばすというプログラムである.

```
#include <stdio.h>

int main(void) {
    int i, N;

    scanf("%d", &N);
    for (i = 0; i < N; i++) {
        if(i % 3 == 0) continue;
        printf("Hello%d¥n", i);
    }

    return 0;
}
```

実行結果は以下のようになる.*15* が入力例.

```
15
Hello1
Hello2
Hello4
Hello5
Hello7
```

```
Hello8
Hello10
Hello11
Hello13
Hello14
```

8. ループ文のいろいろな応用例
   ※量が多いので別の資料に分割します.

9. 変数のスコープ
   ブロックの先頭で宣言された変数は,そのブロックが終わるまで残り続ける.
   なので,例えば for 文中で宣言された変数は,for 文を抜けるとその役目を終え,消える(内部的には,変数宣言によって確保されたメモリ領域が解放される).変数が残り続けられる範囲のことを,変数のスコープ,変数の寿命だとか言ったりする.一般に,使用メモリの節約のため,プログラマは変数のスコープが最小限になるように宣言したほうが良い.
   例を示そう.以下は,入力された N 個の数の総和を求めるプログラムである.

```c
#include <stdio.h>

int main (void) {
    int i, N;
    int sum = 0;

    scanf("%d", &N);
    for(i = 0; i < N; i++) {
        int A;
        scanf("%d", &A);
        sum += A;
    }
    printf("%d¥n", sum);

    return 0;
}
```

   前項に同じ趣旨の例題があったが,こちらは配列の宣言をしていない.
   for 直後のブロック先頭で変数 A を宣言し,そこに scanf で入力した値を入れている.そしてその値を sum に加えていく.for の末尾に達すると,変数 A の寿命は尽きる.しかし次のループで新しく変数 A が生まれる.変数 A は for 外部で使うことはできない.A が生

きることのできる空間は for 内部だけである.

ブロック外で宣言された変数,すなわち int main() {}より前に宣言された変数も作れる.この変数をグローバル変数と呼ぶ.対して,他の変数はローカル変数と呼ばれる.グローバル変数は,そのプログラム内のあらゆる場所でアクセスできる.
グローバル変数はどこからでも利用できる分,その挙動が把握しづらくなるため,あまり使わないほうが良い,というのが一般的な考え方である.
しかし競プロでは,挙動が把握しづらくなるほどの大規模なプログラムを書くことはあまりない.また,

・グローバル変数で宣言した配列は,要素数を大きめに指定できる(理由は割愛)
・初期化宣言をしなくても変数・配列が 0 初期化される

などの理由から,グローバル変数を宣言することを過度に躊躇する必要なないと思われる.