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,…,*AN* 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**  **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 ... *AN*  **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.

・A1, A2, … , AN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**A1からAN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**.

③A1からAN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

・A1からAN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.

1. 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.

|  |
| --- |
| #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.

継続条件

処理

真

偽

while文

1. 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.

|  |
| --- |
| #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.

継続条件

処理

真

偽

do-while文

1. 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.

継続条件

処理

真

偽

for文

更新処理

前処理

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.

|  |
| --- |
| #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.

1. 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.

|  |
| --- |
| #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.

1. 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).

1. 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 |

1. 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.

1. 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.

|  |
| --- |
| #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.