C言語の復習

for 2年生オリエンテーション

鷲崎研究室 修士1年

小林 純一

目次

目次

プログラミングコンテスト

虎の巻

C言語プログラムの雛形

```
#include <stdio.h>
int main(void) {
    int i, n;
    while (1) {
        scanf("%d", &n);
        if (n == 0) {
            break;
        }
        for (i = 0; i < n; i++) {
            printf("Hello, %d\n", i);
        }
    }
    return 0;
}
```

標準入力から
整数nを読み込み、
その数だけ
Hello, (数字)と
標準出力に
出力します

覚えてますか？

C言語プログラムの雛形

```c
#include <stdio.h>
int main(void) {
    int i, n;
    while (1) {
        scanf("%d", &n);
        if (n == 0) {
            break;
        }
        for (i = 0; i < n; i++) {
            printf("Hello, %d\n", i);
        }
    }
    return 0;
}
```

利用している処理

- while文(p.53)
- scanf関数(p.47)
- if文(p.48)
- break文(p.54)
- for文(p.52)
- printf関数(p.45)

プログラミングコンテストとは

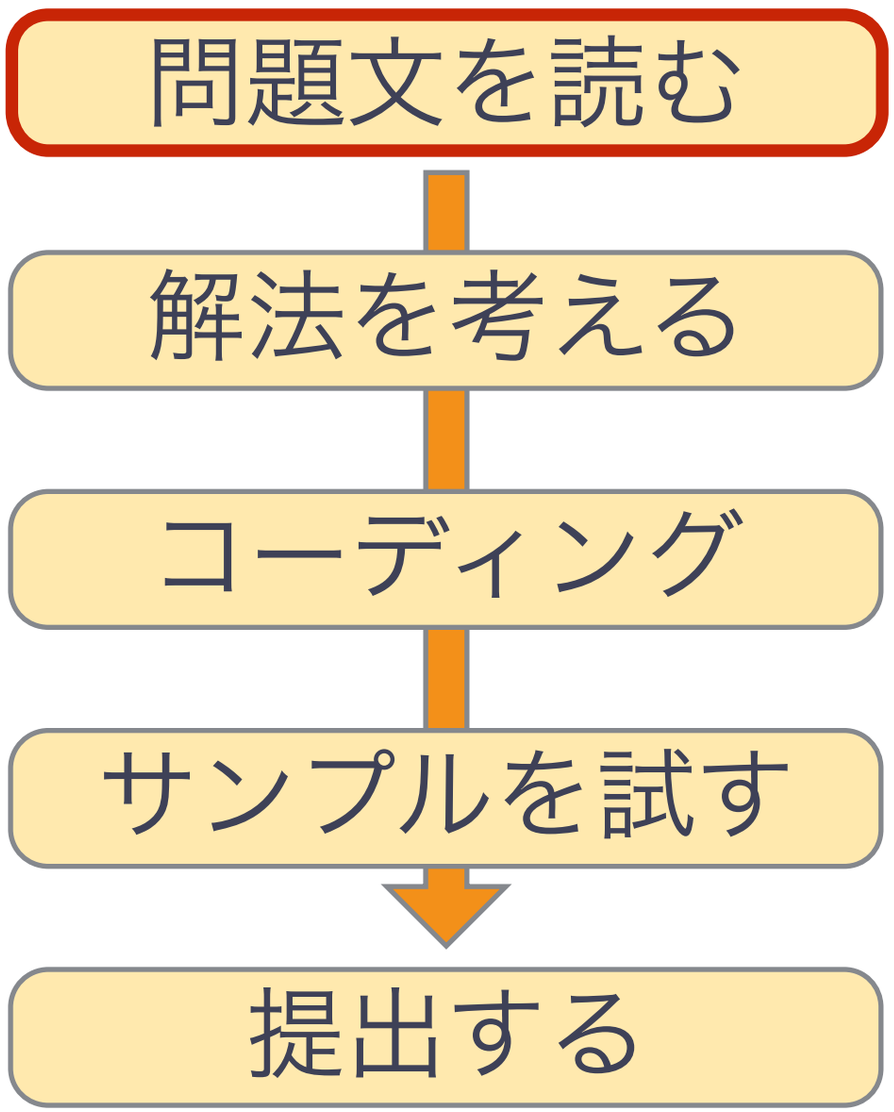
- ある問題に対して**正しく出力するプログラム**を作成する早さを競う
- 厳密には「正しいプログラム」は**必要としない**
  - 出力結果が正答と一致していればよい、ということです
  - 書かれたプログラムが「本当に正しいかどうか」は一種の研究分野でもあります

有名なコンテスト

- **ACM-ICPC**
  - ACMが主催する、大学対抗プログラミングコンテスト
  - **今年は早稲田大学がアジア地区大会のホスト校です**
- **Google Code Jam**
  - Googleが年に1回主催する、オンラインで行われるプログラミングコンテスト

などなど……

コンテストのおおまかな流れ



- まずはここから
- チーム内で分担すると楽です

問題文を読む

- まずは**入出力**を確認する

どんな形式？

入力の制約は？

入力

入力は複数のデータセットから成る。 各データセットは、以下の形式で与えられる。

```
A B
```

A, Bは整数であり、-10000 <= A <= 10000, -10000 <= B <= 10000を満たす。
入力の終了は、"0 0"と書かれた1行によって示される ("入出力の例"を参照せよ)。

出力

各データセットに対して、A+Bを1行ずつ出力せよ。
なお、入力の終了を示す"0 0"に対しては何も出力しないこと。

何を出力する？

問題文を読む

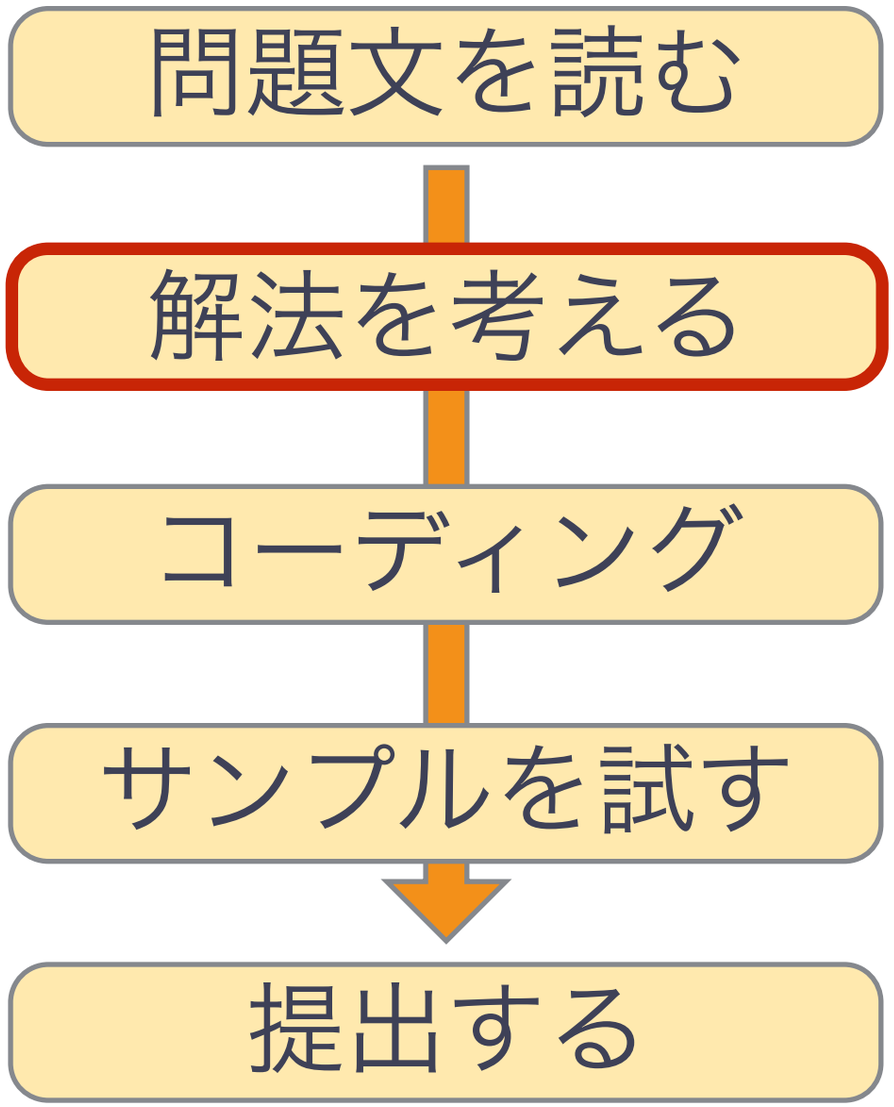
- 入力形式
  - **整数なら整数、文字なら文字**で読み込みましょう
- 入力制限
  - 整数なら値の大きさによって**用意する変数の型**を考えましょう
  - 文字列なら**配列が必要**ですね
- 出力形式
  - **何を出力するのか**をよく読みましょう
  - 文字列なら**大文字と小文字**にも気をつけましょう

問題文を読む

- ストーリーの理解
  - 正直**後回しでいい**です (解答には全く関係しないし)
  - ちょっとしたネタを入れたりしてるので、出題者としては読んでもらえると嬉しいけど……

ちょっとしたネタの例

コンテストのおおまかな流れ



- なんとなく
  「こんな感じかな？」
  みたいな解法は
  思いつくと思います
  それを詰めてみましょう

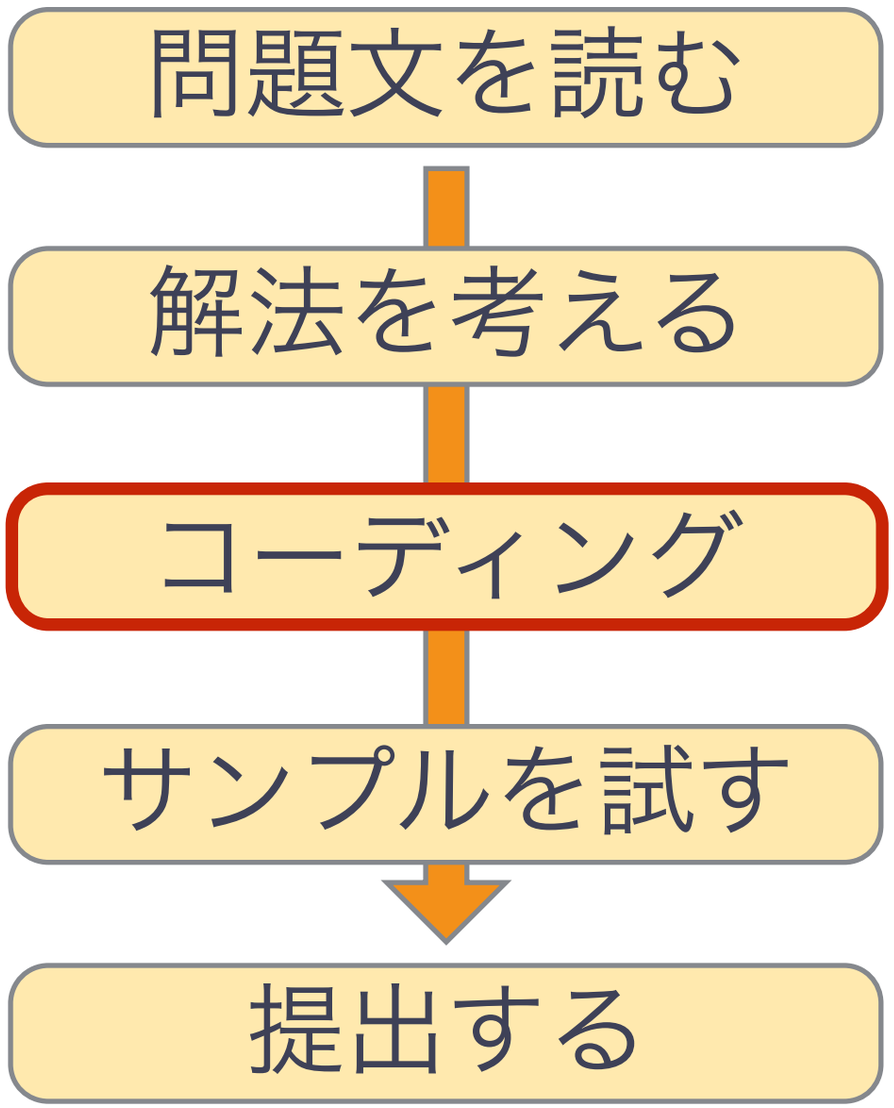
解法を考える

- 解法の流れを考える
  - **紙とペン**を用意しましょう
  - まずAをする、得られた結果からBをする……と、**ある程度具体的な流れ**を考えます
- 最初は無理に**プログラムを意識しなくてもいい**です
- だんだん形になってきたら**チームメイトに説明**してみましょう

解法を考える

- すぐには実装に入らない
  - これ**重要**です
  - 「書きながら考える」のはかえって**余計に時間がかかります**
  - この時点でしっかり解法を固めておくと、
    コーディングがとても**楽になります**

コンテストのおおまかな流れ

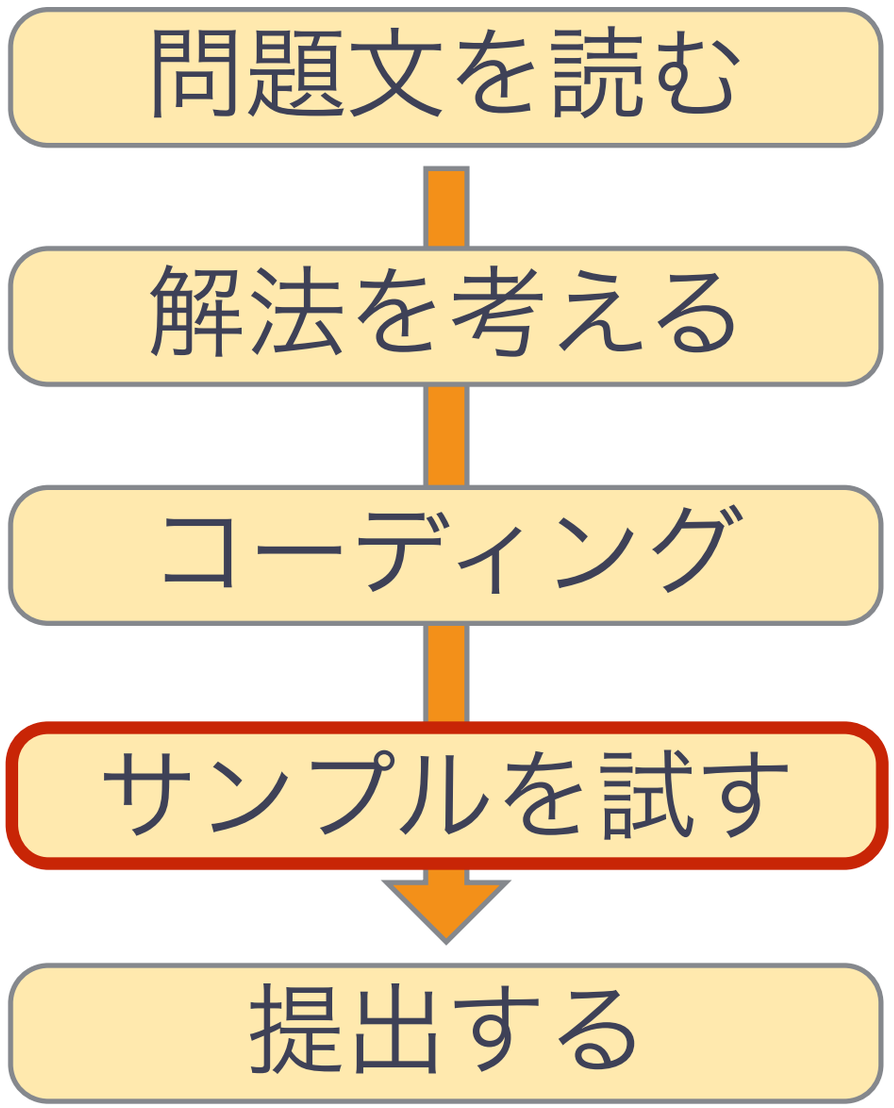


- 解法が固まったら、実際にコードを書きます
- いくつか気をつけるべき点があります

コーディング

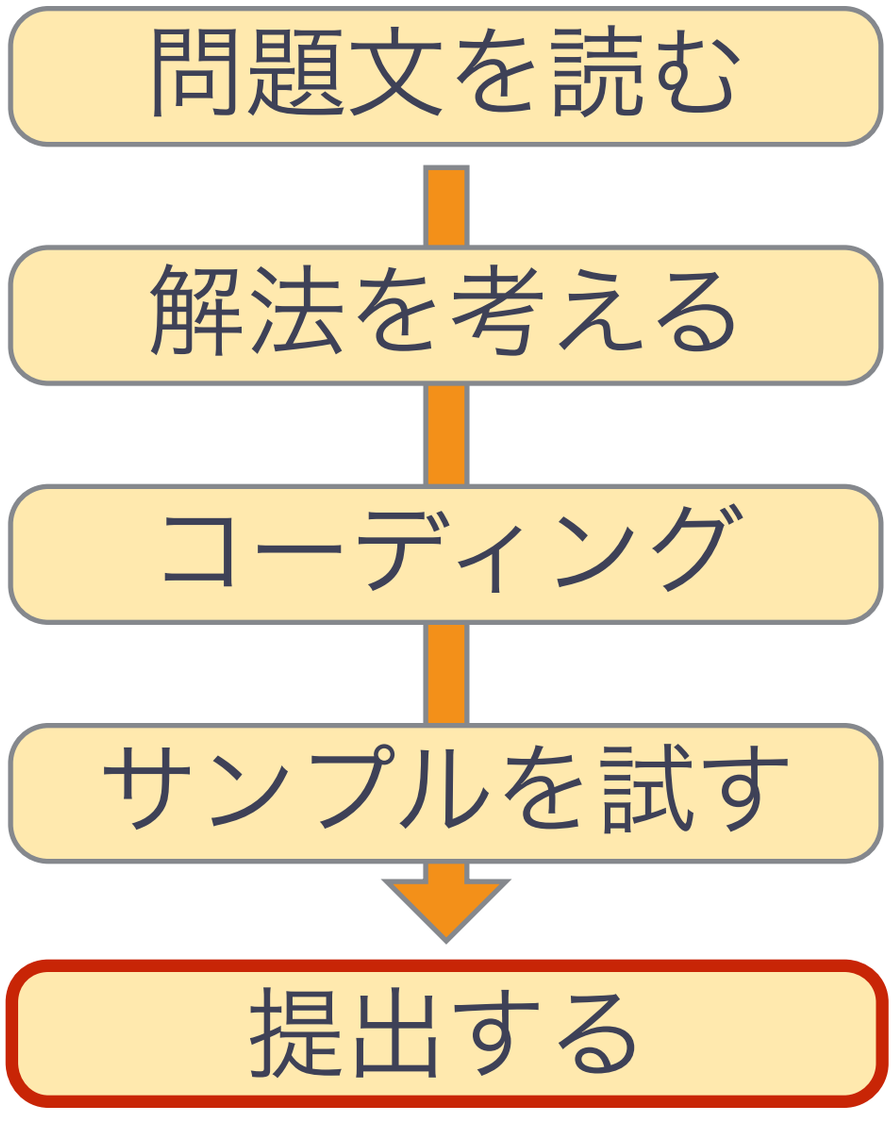
- 入出力制限
  - 「問題文を読む」でも書きましたが、
    入出力の形式には気をつけましょう
- 間違いに備える
  - もし間違っていたときにデバッグしやすいように、
    なるべく**読みやすいコード**を書きましょう
  - コメントを書く必要まではないかもしれませんが、
    インデントはぜひしっかり行ってください
  - (TAさんに見てもらうときもそのほうがいいです)

コンテストのおおまかな流れ



- 一応のプログラムが
  できたら、
  サンプルとして書かれた
  入力を試してみましょう
- もしサンプルで
  問題がなければ、
  実際の入力を入れて
  出力を提出します

コンテストのおおまかな流れ



- Correct! と表示されたらOKです
- 思わぬ落とし穴にハマることも……

提出する

- リダイレクト
  - プログラムの実行時に < をつけてファイルを指定する
    - そのファイルの内容を入力として読み取ります
  - プログラムの実行時に > をつけてファイルを指定する
    - 標準出力ではなく、そのファイルに内容を出力します
    - ファイルが存在しない場合は新規作成されます

```
$ ./a.out < input.txt > output.txt
```

出力はoutput.txtに書き出す

a.outを実行

入力はinput.txtから読み取る

おかしいな？ と思ったら

- よくある失敗集
  - **大文字と小文字**が区別されていない

```
#InClude <stdio.H>
int Main(void) {
    INT i, n;
    foR (i = 0; i < n; i++) {
        prInTF("Hello, %d\n", i);
    }
    Return 0;
}
```

さすがにここまではひどくないと思うけど……

おかしいな？ と思ったら

- よくある失敗集
  - 行末の**セミコロン**を忘れている

```
#include <stdio.h>
int main(void) {
    int i, n;
    for (i = 0; i < n; i++) {
        printf("Hello, %d\n", i)
    }
    return 0;
}
```

コンパイルエラーになります(エラー集も見てね)

おかしいな？ と思ったら

- よくある失敗集
  - **全角文字**を使っている

```
#include <stdio.h>
int main(void) {
    int i, n;
    for (i = 0; i < n; i++)  {
        printf("Hello, %d\n", i);
    }
    return 0;
}
```

特に**スペース**とか、**セミコロン**とか、**括弧**とか!!

おかしいな？ と思ったら

- **エラーメッセージ**を読んでみよう
  - デバッグの参考になります

main関数の中

```
$ gcc foo.c
foo.c: In function 'main':
foo.c:3: 'i' undeclared (first use in this function)
foo.c:3: (Each undeclared identifier is reported only once
foo.c:3:for each function it appears in.)
```

変数iが宣言されていない

foo.cの3行目 (この行数表示があまりあてにならないエラーもあります)

次ページからは**主なエラーメッセージ集**です

おかしいな？ と思ったら

- 'i' undeclared (first use in this function)
  - 訳: 'i' が宣言されていません (この関数での最初の使用です)

```
#include <stdio.h>
int main(void) {
    int n;
    for (i = 0; i < n; i++) {
        printf("Hello, %d\n", i);
    }
    return 0;
}
```

このiが**宣言されていない**

おかしいな？ と思ったら

- parse error at end of input
  - 訳: 入力の最後で構文エラーです

```
#include <stdio.h>
int main(void) {
    int i, n;
    for (i = 0; i < n; i++) {
        printf("Hello, %d\n", i);

    return 0;
}
```

最後に **} が足りない**

おかしいな？ と思ったら

- parse error before '}'
  - 訳: '}' の前で構文エラーです

```
#include <stdio.h>
int main(void) {
    int i, n;
    for (i = 0; i < n; i++) {
        printf("Hello, %d\n", i)
    }
    return 0;
}
```

セミコロンのつけ忘れ

おかしいな？ と思ったら

- parse error before character 0241
  - 訳: 文字0241の前で構文エラーです

この数字は**文字コード**なので
必ずしも0241とは限りません

```
#include <stdio.h>
int main(void) {
    int i, n;
    for (i = 0; i < n; i++)  {
        printf("Hello, %d\n", i);
    }
    return 0;
}
```

全角文字を使っている

おかしいな？ と思ったら

- unterminated string or character constant
  - 訳: 文字列か文字定数が終わっていません

```
#include <stdio.h>
int main(void) {
    int i, n;
    for (i = 0; i < n; i++) {
        printf("Hello, %d\n, i);
    }
    return 0;
}
```

ダブルクォーテーションが閉じられていない

おかしいな？ と思ったら

- undefined reference to 'prnitf'
  - 訳: 'prnitf' というリファレンスは定義されていません

```
#include <stdio.h>
int main(void) {
    int i, n;
    for (i = 0; i < n; i++) {
        prnitf("Hello, %d\n", i);
    }
    return 0;
}
```

prnitf???

おかしいな？ と思ったら

- Segmentation Fault
  - 訳: セグメント例外
    - C言語**最大の敵**です
    - **不正なメモリアクセス**をすると出現します
    - コンパイル時ではなく、**実行時に発生**するエラーです
    - **配列**や**ポインタ**を使っている場合は大抵その辺が怪しいです
    - 使っていない場合は**scanfの&があるか**を確認しましょう
    - もし分からない場合は、一度**放置**するのも一つの手段です

おかしいな？ と思ったら

- No such file or directory
  - 訳: そのようなファイルやディレクトリはありません
    - **ファイル名を指定**したときに起こるエラーです
    - ファイル名を**間違えて入力**している場合が99%以上です
    - 特に**リダイレクトで起こりやすい**のでよく確認しましょう

おかしいな？ と思ったら

- no newline at end of file
  - 訳: ファイルの終わりに改行がありません
    - これはエラーではなく**ワーニング(警告)**です
    - 解決しなくても**問題なく動きます**
    - 示されている通り、ファイルの終わりに**改行がない**のが原因です

C言語クイックリファレンス

定数と変数

- 定数の種類とその例

型	説明	例
数値	数を表す 整数と浮動小数点数がある	100,1.20,1.2e-12など
文字	文字1つを表す '(シングルクォーテーション)でかこって表す	'a', 'x'など
文字列	0文字以上の文字を表す "(ダブルクォーテーション)でかこって表す	"abc", "waseda", ""など

定数と変数

- 変数の型とその種類
  - 主に利用するのは **int** / **char** / **double** だと思います

型	型指定	型の説明	値の範囲
整数	int	整数	-2147483648～2147483647
倍長整数	long long	大きな数の整数	-9.2e+18～9.2e+18
文字	char	1文字	(数値としては)-128～127
単精度浮動小数点	float	小数	3.4e-38～3.4e+38
倍精度浮動小数点	double	より正確な小数	1.7e-308～1.7e+308

配列

- 同じ型の複数のデータを**まとめて扱う**ことができる

```
int a[5];
// int型の配列で、名前はa、要素数は5個

int b[] = {10, 5, 7};
// int型の配列で、名前はb、要素は10,5,7
// int b[3] = {10, 5, 7}; と書いてもよい

int num = b[0];
// b[0] は配列bの0番目の要素を表している
// 配列の添字は0から始まるので注意!!
```

文字と文字列

- 文字(char)型
  - **1バイト(半角)文字**を1文字保持できる変数
    - 1バイトなので、漢字などの**2バイト(全角)文字は扱えません**
- 内部の値は整数であり、**ASCIIコード**と呼ばれる
  - 興味のある人は暇なときにググってください……

```
char c = 'A';
// char型の変数で、名前はc、値は'A'
```

文字と文字列

- 文字列
  - **char型の配列**として表現
  - 文字列の最後には**必ず '\0' が入る**
  - 忘れると**実行時にエラーが出る**ので注意しましょう

```
char s[] = "ABC";
// char型の配列(文字列)で、名前はs、値は "ABC"
// この場合はコンパイラが自動的に末尾に '\0' をつけてくれます

char h[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
// char型の配列で、名前はh、値は "Hello"
// '¥0' を忘れないように!!
```

数値の演算

- 代入
  - 型が同じ変数に値を入れる操作

```
a = 2;
// a に 2 を代入

a = b;
// a に b を代入

a = 2 + 3;
// a に 2 + 3 (= 5) を代入

a = a + 2;
// a に a + 2を代入 ( a自身の値を 2 増やす)
```

数値の演算

- 四則演算子とその意味

演算	演算子	例	意味
加算	+	a+b	aとbを足した値
減算	-	a-b	aからbを引いた値
乗算	*	a*b	aにbを掛けた値
除算	/	a/b	aをbで割った値
剰余	%	a%b	aをbで割った余り

数値の演算

- インクリメント(++)、デクリメント(--)
  - 変数の値を**1増やす**(**1減らす**)
    - a = a + 1; と同じ意味
  - 前につけるか後ろにつけるかで**意味が変わる**

```
int a = 1; int b = a++;
// aの値は最初1で、bに代入する段階で2になる
// bに代入した後にaの値を1増やすので、bには1が代入される

int a = 2; int b = --a;
// aの値は最初2で、bに代入する段階で1になる
// bに代入する前にaの値を1減らすので、bには1が代入される
```

型変換

- 暗黙的型変換
  - 型が違う変数どうしを計算する場合、
    計算結果は**優先度の高い方に統一**される

低		優先度		高
char	int	long	float	double

型変換

- 明示的型変換(キャスト)
  - 変数の値を**強制的に別の型として扱う**場合、
    変数の前に**(型名)**をつける

```
int a = 1, b = 2;
double c = a / b;
// これはNG
// a も b も int であるため、計算結果が0になってしまう

double c = (double)a / b;
// これはOK
// キャストされるため、計算結果は0.5になる
```

標準入出力

- 標準出力(printf()関数)
  - 標準出力(コンソール)に文字を出力する

```
printf("mojiretsu");
// mojiretsuと出力

int a = 4; printf("%d years", a);
// 4 yearsと出力

char[] s = "Hello"; printf("%s", s);
// Helloと出力

printf("This is a pen.\n");
// This is a pen. と出力し、改行する

// 改行は "\n" で表します
```

標準入出力

- 変換指定文字

文字	意味	データ型
%c	1文字出力	char
%d	10進整数出力	int
%ld	10進整数出力	long
%f	小数出力	float, double
%e	指数形式出力	float, double
%s	文字列出力	文字列

%f形式は"%2.3f"のように**桁数指定も可能**
(整数部分2桁、小数部分3桁)

標準入出力

- 標準入力(scanf()関数)
  - 標準出力(コンソール)から文字を読み取る

```
scanf("%d", &a);
// 整数を読み込み、aに格納
// aは整数型である必要があります
scanf("%s", c);
// 文字列を読み込み、cに格納
// cはchar[](文字列)である必要があります
scanf("%d %f %d", &a, &b, &c);
// 半角スペース1文字区切りで整数、小数、整数を読み込む
// a、b、cの型は分かりますよね？
```

条件分岐

- if 文
  - 条件が**満たされるとき**、続く括弧内の処理を行う

```
scanf("%d %d", &a, &b);
if (a == b) {
    printf("a == b");
}
// aがbと等しいなら、a == bと表示する
```

条件分岐

- else 文
  - if 文とセットで、**条件が満たされないとき**の処理を記述する

```
scanf("%d %d", &a, &b);
if (a == b) {
    printf("a == b");
} else {
    printf("a != b");
}
// aがbと等しいなら、a == bと表示する
// そうでないなら、a != bと表示する
```

条件分岐

- switch文
  - ある変数の値によって、**複数の分岐**を行う
  - **case文**によって分岐させる
    - どれにも当てはまらない処理は**defaultに記述**
    - 終わりに**break文**を入れること
      - 入れないと予期しない動作をします
      - なぜ入れるのか気になる人はググってみてね

条件分岐

```
int a;
scanf("%d", &a);
switch(a) {
case 1:
    printf("Taro");
    break;
case 2:
    printf("Jiro");
    break;
default:
    printf("Hanako");
    break;
}
// aの値を読み込み、その値によって表示する人名を変える
```

ループ(繰り返し)

- for 文
  - **決められた回数**だけブロック内の処理を繰り返す
  - 厳密には正しい説明ではないですが……

```
int i;
for (i = 0; i < 10; i++) {
    printf("%d\n", i);
}
// 0から9までを出力し、1つごとに改行する
```

ループ(繰り返し)

- while 文
  - 続く**条件式が正である間**、ブロック内の処理を繰り返す

```
int i = 0;
while (i < 10) {
    printf("%d\n", i);
    i++;
}
// 0から9までを出力し、1つごとに改行する
// 前ページと同じ処理です
```

特殊処理

- break 文
  - その**ブロックを抜けて**次の処理を行う
    - switch文の**各caseの終わり**や、**ループの中**に入れて使う

- continue 文
  - それ以降の処理を行わず、**ループの終わりと判断**する
  - もし**条件式が満たされていなければ、ループの最初に戻る**
    - 主に**ループの中**で使う