C言語入学式
Part 2. ジャンケンで学ぶC言語の基礎

変数とデジタル表現,変数宣言,入出力

変数とデジタル表現

- コンピュータは数字しか理解できません
- ジャンケンは、グー、チョキ、パーですよね？ どうすればよいでしょうか？
- 答えは「**プログラムを書く人が約束事を(人為的に)決める**」です
- この課題では、 **グーは０、チョキは１、パーは２**と決めます

表現	ジャンケン	備考
0	グー	C言語では0から数え始める習わし
1	チョキ	
2	パー	

(脚注) 数字であれば、どう決めてもよいので、 グー１、チョキ２、パー３とか グー１、チョキ２、パー４など、どう決めてもOKです。 ただ、０、１、２にすると後で便利なことがあります

変数の種類(型)

- みなさんがディスプレイで見ているもの(出力)、 キーボードから入力するものは**文字**です
  - コンピュータにとって、文字は、あつかいづらいのです
- あつかいやすいように、**コンピュータの内部表現へ変換**します
- 内部表現として、**値のいれもの＝変数**には、いろいろな種類 (一般には「型」(type)と呼ぶ)が定義されています。 これらを、きちんと指定しないと、正しく動きません
- C言語の型の例(本クラスで主戦場になる型)
  - 整数 = **int型** (整数: integer の int)
  - 整数配列 = **int型の配列** (配列にも型指定が必要です)

(脚注1) 型指定があると、 コンパイルする前に、 ソースコードの間違いを発見できます

(脚注2) きちんと型を書くのが面倒なので、 最近ではPythonのような型を自動推定するスクリプト言語のほうが楽と好まれたりします。ただし、そのために、 速度が犠牲になり、バグ(型の推定ミス、プログラマの読み違え)も誘導します

入出力

- みなさんがディスプレイで見ているもの(出力)、 キーボードから入力するものは**文字**です
- 前述のように文字は扱いづらいのです。 コンピュータが操作しやすいように、 内部表現の値のいれもの＝**変数**には、 いろいろな種類があります。 入出力では、それを正しく指定しないといけません
  - scanfやprintfにあるformat文(先頭の引数`"%何か～"の形式`のこと)が**型の指定**です
  - 変数(型)は`%～`部分のこと。 format文には型以外のメッセージや改行コードなども追加できます

[**C言語の例**]

```c
// キーボードから、整数の変数 jibun に値を代入(この例では \n 不要です(脚注))
scanf("%d", &jibun);

// 整数の変数 jibun を出力 = ディスプレイに表示; \n は改行を意味します
printf("jibun = %d\n", jibun);
```

(脚注) 本テキストであつかっているジャンケンやモグラたたきでは整数変数を読みこんでいます。 この場合、 数字の前後のスペースやENTER叩いた分は適当に無視されて数字だけ読みこむ動作をしますが、 一般にはそうではありません。 scanfは多機能すぎて変な関数なので、 あまり推奨しない方がいいと思うのですが、 ジャンケンやモグラたたきで練習するくらいならscanfが便利です

課題110

```c
// (1) 最初の行に必ず書く呪文

int main () {
  // (2) 整数変数 jibun aite を宣言

  // (3) 代入: jibun に 2、aite に 1 を代入

  // (4) 出力: 変数の値を出力してください
  //   出力例: jibun = 2, aite = 1

  return(0);
}
```

- 整数の変数を2つ宣言してください
- 変数名は分かりやすくローマ字を使います
  - 自分の手は**変数 jibun**
  - コンピュータの手は**変数 aite**
- **変数に値を代入し、その変数の値を出力してください**(今回ジャンケンの手は決め打ちです)
  - ここでは、**コンピュータはチョキ、自分はパーを出したとします**
    - グーは０、チョキは１、パーは２
  - これではゲームらしくありませんが、 まずはここからです。 ソースコードに手を直接書きこむところからやってみてください

ヒント: デジタル表現 変数 出力

課題120

```
int main () {
  // (2) 整数変数 jibun aite を宣言

  // (3) 代入: aite に 1

  // (4) 入力: キーボードから jibun に 2 を代入

  // (5) 出力: 変数の値を出力してください
  //   出力例: jibun = 2, aite = 1
}
```

- 課題110を改造し、キーボードから自分の手を入力できるようにしてください

[実行例]

```
... 120.c ファイルを編集しています(省略) ...

$ cc 120.c
$ ./a.out
2
jibun = 2, aite = 1
```

ヒント: 入力

注意: わずらわしいので、 このあとのテキストでは「(1)いつもの呪文」と最後の「return(0;)」を省略することにします

なお、いちおうワークシートの方には入れてあります

課題130

```
int main () {
  // (2) 整数変数 jibun aite kekka を宣言

  // (3) 代入: aite に 1

  // (4) 入力: キーボードから jibun に 2 を代入

  // (5) 判定: 変数 kekka に判定結果を代入

  // (6) 出力: 変数群の値を出力
  //   出力例: jibun = 2, aite = 1, kekka = 4
}
```

- ジャンケンの判定は次の式で計算できます。ためしてみてください

```
3 + 自分の手 - 相手の手
```

  例: 自分がパー、相手がチョキの場合 3 + 2 - 1 = 4
  - 0 = アイコ
  - 1と4 = (jibunつまりユーザの)負け
  - 2と5 = (ユーザの)勝ち
- 判定結果を変数 kekka に入れ、 jibun aite kekka の3変数の値を出力してください

ヒント: 数値演算

課題140

```
int main () {
  // (2) 整数変数 jibun aite kekka を宣言

  // (3) 代入: aite に 1

  // (4) 入力: キーボードから jibun に 2 を代入

  // (5) 判定: 変数 kekka に判定結果を代入

  // (6) 出力: 変数群の値を出力
  //   出力例: jibun = 2, aite = 1, kekka = 1
}
```

- 変数 kekka を、次の式

```
3 + 自分の手 - 相手の手
```

の**値を3で割った余り**で判定すれば、 0がアイコ、1が(jibunつまりユーザの)負け、2が勝ちになります。 試しに計算してみてください
例: 自分がパー、相手がチョキの場合 3 + 2 - 1 = 4 -> 1 (4を3で割った余りは1)

- 判定結果を変数 kekka に入れ、 jibun aite kekka の3変数の値を出力してください

```
[出力例]

jibun = 2, aite = 1, kekka = 1
```

ヒント: 数値演算 剰余

乱数 (コピー＆ペーストしてok)

- こういう使い方をするものです。コピー＆ペーストしてok

```
// ヘッダファイルが必要です
#include <stdio.h>
#include <stdlib.h> // 乱数を使うため
#include <time.h>   // 時間の関数を使うため

// 以下は main 内

srand(time(NULL));  // 乱数の初期化のため一度だけ実行してください。引数には現在時刻(の秒数)を使っています

int x;
x = rand();         // rand関数を実行するたび x には違う値が入ります
                    // 値の範囲は 0〜約21.5億 (2,147,483,647)です
```

(脚注) 裏側では、ある数式を実行しています。 これは、乱数のようにみえる数字の列を生み出す数式です。 初期値が同じ場合、毎回おなじ数字の列を生成します。 そのため初期化のおまじないが必要です

課題190

```
int main () {
  // (2) 整数変数 jibun aite kekka を宣言

  // (3) 乱数の初期化

  // (4) 乱数で aite の値を決める

  // (4) 入力: キーボードから jibun に 2 を代入

  // (5) 判定: 変数 kekka に判定結果を代入

  // (6) 出力: 変数群の値を出力
}
```

- 乱数(前ページ参照)を使い、 コンピュータの手はコンピュータに考えさせてください。
  これで少しはゲームらしくなってきました
- 判定結果を変数 kekka に入れ、 jibun aite kekka の3変数の値を出力してください
- ヒント
  - `rand()` の返す値は0〜約21.5億(前頁を参照)ですが、 aiteの値は 0 1 2 のいずれかです。 さて、どうすれば `rand()` を使ってaiteの値を決めることが出来るでしょうか?
  - **課題140もヒント**です

```
[出力例]
jibun = 2, aite = 1, kekka = 1
```

ヒント: 乱数 剰余

条件文

条件文 (if)

```
if　　（条件1）{
  条件1を満たしたとき実行するコード
}
else if（条件2）{
  条件2を満たしたとき実行するコード
}
else if（条件3）{
  条件3を満たしたとき実行するコード
}
else {
  条件1,2,3を満たさないときに実行するコード
}
```

(脚注) 最初の if 部分は必須なので if だけの条件文(上の例なら最初の 3 行だけの場合)がありえます。 つまり else if と else は無い条件文もあります。 なお else if は何個あってもかまいません。 else if が多くなるようなら switch 構文を使う書き方もありますが、 C言語の switch 構文は罠があるので注意が必要です。 まずはif文だけ習得することを目指しましょう

例: 条件文

```
if      (条件1) {
  条件1を満たしたとき実行するコード
}

if      (条件2) {
  条件2を満たしたとき実行するコード
}
else {
  条件2を満たさないときに実行するコード
}

// [条件の例]
jibun == 1    // 変数の値が 1 と等しい   (数学とは異なりますね)
jibun  > 0    // 変数の値が 0 より大きい (ここだけ数学と同じ)
jibun =< 2    // 変数の値が 2 以下       (数学とは異なりますね)
```

課題210

```
int main () {
  // (2) 整数変数 jibun aite kekka を宣言

  // (3) 乱数の初期化

  // (4) 乱数で aite の値を決める

  // (4) 入力: キーボードから jibun に 2 を代入

  // (5) 判定: 変数 kekka に判定結果を代入

  // (6) 出力: kachi make aiko を出力
}
```

- 課題190の出力を読みやすくします。 具体的には、 出力(左図(6)部分)の際に 0 1 2 ではなく、aiko make kachi とローマ字で表示してください
  - 変数kekkaの値が、 0 = アイコ、1 = (ユーザの)負け、2 = (ユーザの)勝ち (課題140を参照)

[実行例]

```
$ ./a.out
2
jibun = 2, aite = 1, kekka = 1
make
```

ヒント: 条件文

繰り返し文

例: 繰り返し文

```
[書式]

for（初期値 ; 条件 ; 次へ進めるための式）{

}

[典型例] 0から9までの足し算

int i, sum = 0;
for (i = 0; i < 10; i++) {  // for文は本来の3行分が1行に濃縮された書き方
    sum = sum + i;
}
```

- 数学では「1から10まで足す」気がしますが、 C言語の派閥では「0から10未満まで足す」と考えます
  - よく端の分のあつかいで、コードを間違えます(境界条件の間違い、off by one エラー)
  - こういうのは簡単な例で実際に繰り返しの様子を手でやってみるとよいのです

繰り返し文の意訳(1)

[書式]

```
for（初期値 ; 条件 ; 次へ進めるための式）{
  何かの処理
}

初期値;
「条件」が満たされている間は繰りかえす {
    何かの処理
    次へ進めるための式
    // 「条件」が満たされて～行へ戻る
}
```

繰り返し文の意訳(2)

[書式]

```
for（初期値 ; 条件 ; 次へ進めるための式）{
  何かの処理
}

初期値;
「条件」が満たされている間は繰りかえす {
    何かの処理
    次へ進めるための式
    // 「条件」が満たされて〜行へ戻る
}
```

[典型例] 0から9までの足し算

```
int i, sum = 0;
for (i = 0; i < 10; i++) {
    sum = sum + i;
}

// 意訳; これを省略して上のように書きます
i = 0;
「条件( i < 10 )」が満たされている間は繰りかえす {
    sum = sum + i;
    i++;
    // 「条件」が満たされて〜行へ戻る
}
```

繰り返し文の意訳(3)

[典型例] 0から9までの足し算

```
int i, sum = 0;
for (i = 0; i < 10; i++) {  // for文は本来の3行分が1行に濃縮された書き方
    sum = sum + i;
}

// 上の文は、こう読みます。こういうのは簡単な例で実際に繰り返しの様子を手でやってみるとよいのです

i = 0 （初期化の部分は、最初の1回だけ実行されます）
条件文 i < 10 を満たすので { } の中を実行します
sum = sum + i;（つまり sum = sum + 0;）を実行し
i++; を実行して i が 1 になり、また for 文の真ん中へ戻ります(注: 先頭には戻りません)
条件文 i < 10 を満たすので { } の中を実行します
sum = sum + i;（つまり sum = sum + 1;）を実行し
i++; を実行して i が 2 になり、また for 文の真ん中へ戻ります
... 以下略 ...
```

課題310

```
int main () {
  // (A) 一回だけ実行する部分

  // (B) 繰り返し文(for)
  for (                              ) {
     // (C) くりかえし実行する部分




  }

}
```

- ジャンケンを3回できるようにしてください
- まず、 課題210の(2)〜(6)のうち、 1回だけ実行する部分と繰り返す部分がどこかを考えましょう

```
[実行例]

$ ./a.out
2
jibun = 2, aite = 1, kekka = 1
make
1
jibun = 1, aite = 1, kekka = 0
aiko
...略...
```

ヒント: 繰り返し文 (これでジャンケンが一通り完成です)

ジャンケン 発展編

発展課題INDEX

- 410: ジャンケンを何回できるか?(回数の上限を)指定できるようにしてください
- 420: ジャンケンの回数に上限をなくしてください。 ただし数字の9を入れたら終了です
  - つまり無限ループしていて9が入力されたときには終了するという意味
  - キーボードから入力される値は 0 1 2 9 のいずれかという想定
- 430: 終了時に、何回勝ったか?も表示してください
- 440: 終了時に、何勝、何敗、あいこが何回か?を表示してください
- 450: 課題310をwhile文で書き直してみよう
- 460: [お作法] キーボードから入力できる値を確認するように変更してください
  - 具体的には、キーボードから入力される値は 0 1 2 9 のいずれかのみです
  - ジャンケンくらいでは、ほぼ無関係ですが、 どんなときでも入力値は確認するべきで、 それは特にセキュリティの観点で重要です。 ちなみに、こういう確認を input validation (入力の検証)と呼びます

発展課題410: ジャンケンを何回できるか?上限を指定できる

```
int main () {
  //（A）一回だけ実行する部分

  //（B）繰り返し文(for)
  for (                              ) {
     //（C）くりかえし実行する部分




  }

}
```

- **課題310が書けることは大前提です**
- 以下では、 課題310のワークシートの(A)(B)(C)部分の変更点もしくはヒントだけ書いていきます
- (A)部分
  - 変数宣言: 既存の変数に追加して、新変数 jougen を定義してください。 ループできる回数(ループの上限値)を代入する変数です
  - キーボードから何回できるか?(数字)を入力し jougen に代入
  - 乱数の初期化(課題310と同じ)
- (B)くりかえし文では「jougen 回ジャンケンを行える」ようにします (それ以外は課題310と同じ)
- (C)部分の変更は無いはず

発展課題420: ジャンケンの上限なし

```
int main () {
  //（A）一回だけ実行する部分

  //（B）繰り返し文(for)
  for (                              ) {
     //（C）くりかえし実行する部分



  }
```

```
}
```

- **課題310が書けることは大前提です**
- (A) 必要に応じて、変数を削除、新たに宣言してください
- (B) くりかえし文
  - 「無限ループ」にします
- (C)
  - jibun の値が 9 ならループを終了

発展課題430: 終了時に何勝を表示

```
int main () {
  //（A）一回だけ実行する部分

  //（B）繰り返し文(for)
  for (                              ) {
     //（C）くりかえし実行する部分



  }
```

```
}
```

- **課題310が書けることは大前提です**
- (A) 回数を数える変数 kaisuu_kachi を追加
- (B) くりかえし文
- (C)
  - kachi,make,aikoを表示する条件文の中で、条件が「勝ち」のとき kaisuu_kachi を加算していきます
  - 変数 kaisuu_kachi の値を表示します

発展課題440: 終了時に何勝/何敗/あいこの回数を表示

```
int main () {
  // (A) 一回だけ実行する部分

  // (B) 繰り返し文(for)
  for (                              ) {
     // (C) くりかえし実行する部分




  }
```

```
}
```

- ここは、ほぼノーヒントとしましょう。前問430と、ほぼ同じですから
- いずれの箇所も3倍の長さになりますね
  - 変数は3つ必要;勝ち、負け、アイコの回数を記録する
  - 条件文も3ヶ所修正
  - 最後に3つの値を表示する

発展課題450: 課題310のwhile版

- ヒント: 書き方のバリエーションがありえますが、**一番短い書き方をした場合、修正箇所は2行だけ**です