第３章

じゃんけんゲーム

本章では《じゃんけんゲーム》を作成します。最初は単純なものを作り、少しずつ手を加えて機能を追加していきます。

この章で学ぶおもなこと

- switch 文
- char型
- 条件演算子・条件式
- 特定範囲の数値の読込み
- 文字コード
- 漢字コード
  （JIS / シフト JIS / EUC）
- 漢字を含んだ文字列
- ワイド文字
- ポインタによる文字列の走査
- 文字列の配列
  （２次元配列／ポインタの配列）
- 関数
- 識別子の有効範囲
- ⊙wchar_t 型
- ⊙isprint 関数
- ⊙CHAR_BIT
- ⊙CHAR_MAX
- ⊙CHAR_MIN
- ⊙SCHAR_MAX
- ⊙SCHAR_MIN
- ⊙UCHAR_MAX

3-1 じゃんけんゲーム

本章では、二人のプレーヤが対戦する《じゃんけんゲーム》を作成します。もちろん、対戦するプレーヤは、コンピュータと人間です。

基本設計

まずは《じゃんけんゲーム》の大まかな設計をしましょう。プログラムの流れは、次のようにします。

1. コンピュータの手を決定する。
2. 「じゃんけんポン」と表示して、人間が手を入力する。
3. 勝敗の判定を行い、その結果を表示する。
4. 続行するかどうかをたずね、人間が希望すれば1に戻る。

次に、各ステップを少し詳しく設計していきます。

1 コンピュータの手を乱数で決定します（具体的な値は2で設計します）。

人間の手を読み込む2よりも前に行うのは、コンピュータが勝つように作為することを防ぐためです。

▶ **List 3-8**（p.93）では、コンピュータがズルをする《後出しじゃんけん》のプログラムを作成します。

2 手の入力を"グー"、"チョキ"、"パー"の文字列で行わせると、**タイプミス**が起こるかもしれません。たとえば、グーやパーの長音記号をマイナス記号に間違えるといった具合です。

そこで、**Fig.3-1** に示すように、グー、チョキ、パーの手を0，1，2に対応させることにします（型は`int`型とします）。

そうすると、次のように表示した上で選ばせる（数値を入力させる）ことができます。

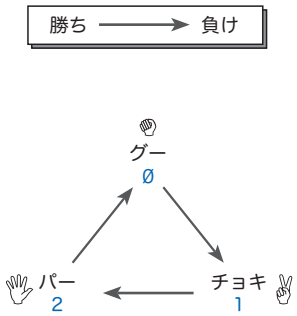
Fig.3-1 手とその値

じゃんけんポン…(0) グー (1) チョキ (2) パー：

人間の手とコンピュータの手を同じ値で表現すれば、一貫性が保てるため好都合です。これで、1の設計で未解決だった手の値も決定です。

3 コンピュータと人間の手から、勝敗を判定します。

ここで、人間とコンピュータの手を表す変数を *human* と *comp* とします。手と勝敗の関係を示したのが **Fig.3-2** です。0, 1, 2, 0, 1, 2,…という循環において、矢印の**始点**側が“勝ち”で、**終点**側が“負け”です。



a 引き分け

human	*comp*	*human* - *comp*
0	0	0
1	1	0
2	2	0

b 人間の勝ち

human	*comp*	*human* - *comp*
0	1	-1
1	2	-1
2	0	2

c 人間の負け

human	*comp*	*human* - *comp*
0	2	-2
1	0	1
2	1	1

Fig.3-2 勝敗の判定

図内に示す各表は、両者の手の値と、*human* から *comp* を引いた値をまとめたものです。

a 引き分け

human と *comp* の値が等しければ“引き分け”です。このとき、*human* - *comp* の値は 0 となります。

b 人間の勝ち

人間が始点でコンピュータが矢印の終点となる組合せが“人間の勝ち”です。このとき、*human* - *comp* の値は -1 または 2 となります。

c 人間の負け

人間が終点でコンピュータが矢印の始点となる組合せが“人間の負け”です。このとき、*human* - *comp* の値は -2 または 1 となります。

三つの判定は、共通の式 (*human* - *comp* + 3) % 3 で行えます。この値が 0 であれば引き分け、1 であれば人間の負け、2 であれば人間の勝ちです。

4 これについて詳しい説明の必要はないでしょう。

switch 文

ここまでの設計をもとに作成したプログラムを **List 3-1** に示します。

List 3-1 chap03/jyanken1.c

```
/* じゃんけんゲーム（その１）*/

#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int human;              /* 人間の手 */
    int comp;               /* コンピュータの手 */
    int judge;              /* 勝敗 */
    int retry;              /* もう一度？ */

    srand(time(NULL));      /* 乱数の種を初期化 */

    printf("じゃんけんゲーム開始!!\n");

    do {
        comp = rand() % 3;      /* コンピュータの手（0〜2）を乱数で生成 */

        printf("\n\aじゃんけんポン…(0)グー (1)チョキ (2)パー：");
        scanf("%d", &human);        /* 人間の手を読み込む */

        printf("私は");             /* コンピュータの手を表示 */
        switch (comp) {
         case 0: printf("グー");    break;
         case 1: printf("チョキ");  break;
         case 2: printf("パー");    break;
        }
        printf("です。\n");

        judge = (human - comp + 3) % 3;     /* 勝敗を判定 */

        switch (judge) {
         case 0: puts("引き分けです。");         break;
         case 1: puts("あなたの負けです。");     break;
         case 2: puts("あなたの勝ちです。");     break;
        }

        printf("もう一度しますか…(0)いいえ (1)はい：");
        scanf("%d", &retry);
    } while (retry == 1);

    return (0);
}
```

switch文

switch文

実行例

```
じゃんけんゲーム開始!!

♪じゃんけんポン…(0)グー (1)チョキ (2)パー：2↵
私はグーです。
あなたの勝ちです。
もう一度しますか…(0)いいえ (1)はい：1↵

♪じゃんけんポン…(0)グー (1)チョキ (2)パー：1↵
私はチョキです。
引き分けです。
もう一度しますか…(0)いいえ (1)はい：0↵
```

まずは、プログラムを実行してみましょう。

手を入力するように促されますので、0，1，2 の値を打ち込むと、勝敗結果が表示されます。もう一度行うかどうかの入力が促されますので、1 を打ち込むと、再びジャンケンを行えるようになっています。

switch文の構文

```
switch（式）文
```

コンピュータの手の表示と判定結果の表示を行っているのが、網かけ部の **switch** 文です。

まず、制御式である**式**が評価されます。そして、**case** に続く値が、評価結果と一致している**ラベル**へと、プログラムの流れが移ります。

ただし、一致するラベルがない場合は、**default** ラベルがあればそこに移動し、なければ **switch** 文を抜け出します。

ラベルに飛んだ後は、置かれている**文**を順次実行します。その過程で、**break** 文に出会うと **switch** 文の実行を終了します。**Fig.3-3** のプログラムと、その流れを示した図を見比べれば理解できるでしょう。

```
switch (sw) {
 case 1  : puts("A");
           puts("B");  break;
 case 2  : puts("C");
 case 5  : puts("D");  break;
 case 6  :
 case 7  : puts("E");  break;
 default : puts("F");  break;
}
```

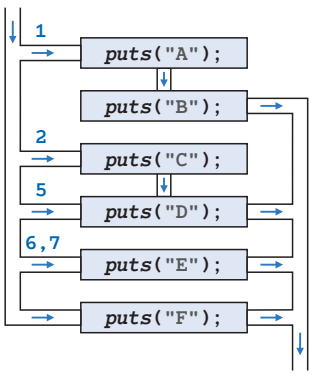


Fig.3-3 switch文によるプログラムの流れの分岐

if 文と **switch** 文のどちらを使っても実現できる分岐は、**switch** 文を利用して実現したほうが読みやすくなる傾向があります。そのことを、以下に示す二つのプログラム部分で考えましょう。

```
if (p == 1)
    c = 15;
else if (p == 2)
    c = 23;
else if (p == 3)
    c = 57;
else if (q == 4)
    c = 84;
```

```
/* 左のif文を書き直したもの */

switch (p) {
 case 1  : c = 15;  break;
 case 2  : c = 23;  break;
 case 3  : c = 57;  break;
 default : if (q == 4) c = 84;
}
```

まずは、**if** 文をじっくり読んでみましょう。先頭三つの **if** は *p* の値を調べ、最後の **if** は *q* の値を調べています。変数 *c* に 84 が代入されるのは、*p* が 1，2，3 のいずれでもなく、かつ *q* が 4 であるときです。

連続した **if** 文において、分岐のための比較対象となるのは、必ずしも単一の式であるとは限りません。最後の判定は、**if**（*p* == 4）と読み間違えられたり、**if**（*p* == 4）の書き間違いではないかと誤解されたりするかもしれません。

その点、**switch** 文のほうは、全体の見通しがよいため、プログラムを読む人が、そのような疑念を抱くことが少なくなります。

<手>を表す文字列

前のプログラムでは、人間が手を入力すると、『私はグーです。』とコンピュータの手が表示されます。『私はグーで、あなたはパーです。』と、コンピュータの手だけでなく**人間の手も表示する**ように変更しましょう。そのプログラムを **List 3-2** に示します。

List 3-2 chap03/jyanken2.c

```
/* じゃんけんゲーム（その２：両者の手を表示）*/

#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int human;              /* 人間の手 */
    int comp;               /* コンピュータの手 */
    int judge;              /* 勝敗 */
    int retry;              /* もう一度？ */

    srand(time(NULL));      /* 乱数の種を初期化 */

    printf("じゃんけんゲーム開始!!\n");

    do {
        comp = rand() % 3;      /* コンピュータの手（0〜2）を乱数で生成 */

        do {
            printf("\n\aじゃんけんポン…(0)グー (1)チョキ (2)パー：");
            scanf("%d", &human);    /* 人間の手を読み込む */
        } while (human < 0 || human > 2);

        printf("私は");

        switch (comp) {             /* コンピュータの手を表示 */
         case 0: printf("グー");    break;
         case 1: printf("チョキ");  break;
         case 2: printf("パー");    break;
        }

        printf("で、あなたは");

        switch (human) {            /* 人間の手を表示 */
         case 0: printf("グー");    break;
         case 1: printf("チョキ");  break;
         case 2: printf("パー");    break;
        }

        printf("です。\n");

        judge = (human - comp + 3) % 3;                 /* 勝敗を判定 */

        switch (judge) {
         case 0: puts("引き分けです。");        break;
         case 1: puts("あなたの負けです。");    break;
         case 2: puts("あなたの勝ちです。");    break;
        }

        printf("もう一度しますか…(0)いいえ (1)はい：");
        scanf("%d", &retry);
    } while (retry == 1);

    return (0);
}
```

1 2 3 ほとんど同じ

実行例

```
じゃんけんゲーム開始!!

♪じゃんけんポン…(0)グー (1)チョキ (2)パー：3↵

♪じゃんけんポン…(0)グー (1)チョキ (2)パー：-1↵

♪じゃんけんポン…(0)グー (1)チョキ (2)パー：2↵
私はグーで、あなたはパーです。
あなたの勝ちです。
もう一度しますか…(0)いいえ (1)はい：1↵

♪じゃんけんポン…(0)グー (1)チョキ (2)パー：2↵
私はチョキで、あなたはパーです。
あなたの負けです。
もう一度しますか…(0)いいえ (1)はい：0↵
```

1 人間の手を読み込む部分です。0，1，2 のみを受け付けるようにするために **do** 文を導入しています。

この **do** 文は、変数 *human* に読み込んだ値が 0 より小さいか、2 より大きいあいだ繰り返されます。そのため、**do** 文が終了した時点での変数 *human* の値は、**必ず 0 以上 2 以下となります。**

なお、この **do** 文は、ド・モルガンの法則（p.21）を用いると、以下のように実現できます。

```
do {
    printf("\n\aじゃんけんポン…(0)グー (1)チョキ (2)パー：");
    scanf("%d", &human);          /* 人間の手を読み込む */
} while (!(human >= 0 && human <= 2));
```

▶ もし *human* の値の妥当性（0，1，2 のいずれかの値になっているかどうか）をチェックしなかったらどうなるでしょう。0，1，2 以外の値が打ち込まれると、人間の手を表示する **3** の **switch** 文は実質的に素通りされます。そのため、『私はチョキで、あなたはです。』と表示されることになります。

2 コンピュータの手を表示する **switch** 文です（前のプログラムと同じです）。

3 人間の手を表示する **switch** 文です（本プログラムで追加しています）。

2 と **3** の **switch** 文は、ほぼ同じです。よく似たコードが繰り返されていることもあって、プログラムが長くなっています。

その上、"グー"、"チョキ"、"パー" が、独立した文字列リテラルとして各二つ、文字列リテラルの一部として各一つと、三度ずつも現れます。

もし、手の表記を〔カタカナ〕から〔ひらがな〕に変えたり、0，1，2 以外の値に変更しようとすると、修正や変更は何箇所にも及ぶことになります。

▶ 手の値の変更については p.92 で検討します。

まとめ

選択文（if 文と switch 文）

単一の式の値によってプログラムの流れを分岐するには、**if** 文よりも **switch** 文のほうが適している（プログラムの意図をつかみやすい）ことが多い。