練習課題

ブラックジャックを作ろう！

卒業制作としてブラックジャックを作ってみましょう。

カードゲーム、いわゆるトランプは、コンソール画面を使うプログラムの練習として、とても相性がいいと思います。その中でブラックジャックを選んだのは、単にわたしがルールを知っているゲームだからです（ぼっちは友だちがいないので、基本的に多人数ゲームのルールとか知りません）。

「作れそうだな！」と思った方は、いきなりチャレンジしてみても結構です。ただ、次の rand 関数とブラックジャックのルールの項だけは読んでおいてください。

ちょっと自信のない方は、いっぺんに全部作る必要はありませんから、必要な部分や関数を徐々に作っていって、最後にくっつけるといいと思います。演習形式で一緒にやっていきましょう。

プログラムの考え方なんてたくさんありますから、「こう作らなければならない」といった模範解答はありませんが、できるだけほかの遊び（大貧民とか）にも転用できるように考えて作っていくといいと思います。

■rand 関数

カードゲームに限らず、ゲームを作る上で避けて通れないのが rand 関数です。ゲームでは偶然の要素を演出する必要があります。でも、コンピュータにシャッフル（カードをまぜこぜする）させると、まぜこぜしているはずなのに、いつも同じ並びになったりするのです！！　ああ、なんてこと！

そこで登場するのが、でたらめな数（疑似乱数）を作り出すための関数、rand です。引数は必要なく、戻り値として 0～32767 の間の任意の整数を返します。つまり、

```
a = rand();
```

と書いておくと、変数 a に 0～32767 の間のでたらめな数を入れてくれます。もちろん、変数 a は上記の範囲の整数を代入できる型でなくてはなりません。

「そんなに範囲が広いと困る」という場合は、欲しい範囲で割った余りを使いましょう。たとえば、今回はカードをシャッフルするのに乱数を使いたいのですが、カードは 52 枚しかないため、0～32767 の範囲の乱数などいりません。

```
a=rand()%52;
```

とすれば、0～32767 の間のでたらめな数を 52 で割った余り、すなわち 0～51 の間のでたらめな数が得られます。

コンピュータは生真面目なので、でたらめな数を作るのがとっても苦手です。rand 関数もきちんとした手順を踏んで「でたらめっぽい数」を作っているだけなので、元になる数（シード：種といいます）を変えてあげないと、いつも同じ並びのでたらめな数が出てきてしまいます。だから「疑似」乱数と言うのですね。

この問題を解決するには、シードを変更すればよいのですが、シードに規則性が出てしまうといけません。利用者に入力してもらうのも手ですが、利用者が同じシードを入力してしまうと、いつも同じカードの並びになってしまいます。

そこでよく使われるのが、システム時間をシードにする方法です。rand 関数を使う前に次のような文を書いておきましょう。

```
srand(time(NULL));
```

この辺は今の時点では詳細に知っていなくても大丈夫ですが、簡単に触れておくと、シードを変更するための srand 関数に、引数としてシステム時間を得る time 関数を与えています。これならプレイした時刻の違いで、異なる乱数を作り出すことができます。

なお、srand の引数は unsigned int 型なので、キャストしておくと安全です。

```
srand((unsigned)time(NULL));
```

rand 関数、srand 関数を使うためには stdlib.h、time 関数を使うには time.h というヘッダファイルをインクルードする必要があります。本書でさんざん使った stdio.h では使えないので、プログラムの先頭に次の文を足してください。

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

もちろん、ブラックジャックでは printf 関数なども使いますから、stdio.h をインクルードすることも忘れないでくださいね。

■ブラックジャックのルール

なんだかいろいろなローカルルールがあって、よくわかりません。カジノに行っても、その土地、そのハウスごとに特殊ルールがありますし。

なので、思いっきり簡単なところだけつまみ食いしてみました。今回の練習用ブラックジャックはこのルールで作ってみましょう。もちろん、自信のある方や、一度このルールでプログラムを完成させた方、ブラックジャックに詳しくてこんないんちきルールでは納得できない方は、別のルールに対応したプログラムも作ってみてください。

＜本書のルール＞

- プレイヤーとディーラーで 1 対 1 勝負
- 絵札は 10 として数える
- A は最初 11 として数えるが、バースト（カードの合計点が 21 を超えちゃった）したときは、1 に数え直していい
- 手札はプレイヤー、ディーラーともに常にオープン
- 手札はプレイヤー、ディーラーに最初に 2 枚ずつ配り、その後追加で引くかどうかは、プレイヤー、ディーラーの選択に任される
- 先にカードを引くのはプレイヤー
- カードの合計点が 21 以内で、相手より大きな値である側が勝ち
- プレイヤーはカードの合計点が 21 を超えると負け
- ディーラーはカードの合計点が 21 を超え、かつプレイヤーの合計点が 21 以内だと負け
- カードの合計点が同じ、あるいはプレイヤーもディーラーもカードの合計点が 21 を超えたときは引き分け
- ディーラーの思考方法は難しいので、プレイヤーの手札の状態とか全然考えないラスベガススタイルにする。16 以下なら必ず次のカードを引くし、17 以上ならもう引かない
- 使うカードは 1 組（1 デッキ）だけ。1 勝負ごとにシャッフルする

■練習 1

カード情報を格納する構造体を作ってみましょう！

■練習 2

構造体 card にカード情報を代入してみましょう。今の時点では、メンバ value と disp は同じ値（絵札は、11、12、13 と数える）で構いません。

■練習 3

このままではいつも同じ並びのカード（引くと、1、2、3、4、5、……と出てくる）です。カードを 100 回シャッフルしてみましょう。

■練習 4

カードを表示する関数 disp を作ってみましょう。引数は、今プレイヤー（ディーラー）が何枚カードを持っているかと、プレイヤーの持ちカード全体の情報です。戻り値はいりません。

■練習 5

カードの値をもらうと検査し、絵札なら値を 10 に、A なら 11 に変更する関数 num を作ってください。

■練習 6

ブラックジャックで使うであろう変数を予想して、変数を宣言してみましょう。何と何が必要ですか？

■練習 7

勝ち負けを判定する関数 hantei を作ってみましょう。引数は 2 つ、プレイヤーの合計点とディーラーの合計点です。戻り値として整数をセットし、勝ち負けの状態に応じて以下を返すようにします。

- 1…ディーラーの勝ち
- 2…プレイヤーの勝ち
- 3…引き分け

余裕があれば勝ち負けの表示も行なってみてください。ここでは、文字列「プレイヤー」にはポインタ ps が、文字列「ディーラー」にはポインタ ds がセットされています。

■練習 8

プレイヤーとディーラーに手札を 2 枚ずつ配ってみましょう。

■練習 9

A は最初 11 と数えていましたが、合計点が 21 を突破してしまいました！　これでは負けです。手札に A があれば 11 を 1 に数え直して状況を立て直しましょう！

■練習 10

あとは、「もう 1 枚カードを引く？　もう 1 枚カードを引く？」という部分を作り込んであげれば、今までに作ったパーツの組み合わせでブラックジャックのプログラムが作れるのではないでしょうか。

「もう引かないよ」となった場合の処理を、忘れずに記述してあげるのがポイントです。うんと時間がかかっていいですから、いろいろ試行錯誤してみてください。解答例を見るのは考えに考え抜いた後でも遅くありません。是非、解答例を上回る美しいプログラムを書いてみてください。

※解答例は別ファイル（makeBJ_A.pdf）にまとめています。