# プログラミング言語実験
# 第3回
# コンピュータ大貧民
— クライアントの基本機能の実装（カードの提出） —

2019年04月22日、23日

## 1　カードの提出

### 1.1　配列とカードの表現

UECdaにおいて、場に出ているカードや手持ちのカードは、$8 \times 15$の配列として与えられます。概要を表1に示します。0行目〜4行目でカードを表現し、5行目〜7行目で革命中等の場の状態を表現します。詳細はUECdaのマニュアル
（http://www.tnlab.inf.uec.ac.jp/daihinmin/2018/document.html）を見てください。
UECdaにおけるクライアントプログラムを作成するには、“配列を読み取り、どのカードを出すかを決定し、やはり配列に値をセットして送り返す”プログラムを書くということになります。

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | スペード、左から3,4,5,...,Q,K,A,2 | | | | | | | | | | | | | |
| 1 | | ハート | | | | | | | | | | | | | |
| 2 | | ダイヤ | | | | | | | | | | | | | |
| 3 | | クローバー | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | | | |
| 7 | | | | | | | | | | | | | | | |

表1: コンピュータ大貧民におけるカード表現

配列中、表中で持っているカードには1が、持っていないカードには0が入ります。ジョーカーの場合には、特別に2が入ります。4行目、0列目、14列目は、ジョーカーを使った特殊な手（5カードやA, 2, ジョーカーの階段等）を表現するのに使われます。

次に、いくつかの例を示します。この例では、**0が入っている部分は空白**にしてあります。また、**5行目以降は省略**してあります。

- 手持ちのカードがダイヤの 3, 4、ハートの 9 と 11 の場合：

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | |
| 1 | | | | | | | | 1 | | 1 | | | | | |
| 2 | | 1 | 1 | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | |

- 手持ちのカードがダイヤの 3, 4、ハートの 9 と 11 とジョーカーの場合：

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | |
| 1 | | | | | | | | 1 | | 1 | | | | | |
| 2 | | 1 | 1 | | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | | | |
| 4 | | 2 | | | | | | | | | | | | | |

自分の手持ちのカードとしてジョーカーがある場合は、配列の [4, 1] に 2 が入ります。

- 場のカードがクローバーの 4, 5, 6 の場合：

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | |
| 3 | | | 1 | 1 | 1 | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | |

- 場のカードがクローバーの 4, 5, 6 で、ジョーカーがクローバーの 6 として代用された場合：

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | | |
| 2 | | | | | | | | | | | | | | | |
| 3 | | | 1 | 1 | 2 | | | | | | | | | | |
| 4 | | | | | | | | | | | | | | | |

ジョーカーが代用された場合は、代用の対象となったカードの部分に 2 が入ります。

- 場のカードが 7 の 5 カードの場合：

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | 1 | | | | | | | | | |
| 1 | | | | | | 1 | | | | | | | | | |
| 2 | | | | | | 1 | | | | | | | | | |
| 3 | | | | | | 1 | | | | | | | | | |
| 4 | | | | | | 2 | | | | | | | | | |

5 カードや、A, 2, ジョーカーの階段を表現する場合、4 行目や 0 列目、14 列目が使用されます。

## 1.2 提出するカードを選択する関数 select_submit_cards

自分がカードを提出する番が回ってきたとき、関数 select_submit_cards が呼ばれます。この関数は select_cards.c に書かれています。プロトタイプ宣言は次の通りです。

Listing 1: 関数 select_submit_cards のプロトタイプ宣言

```
1 void select_submit_cards(int out_cards[8][15],int my_cards[8][15],
2                          state *field_status);
```

それぞれの引数は、次のような役割です。

- int out_cards[8][15]：
  提出する手を格納する配列です。
- int my_cards[8][15]：
  自分のカードが格納されている配列です。
- state *field_status：
  場の情報が格納されている構造体です。メンバ cards が場に出ているカードの配列です。他の情報も格納されています。詳しくは common.h を見てください。

結局のところ、この講義での「クライアントが提出するカードを選択する」とは、**「my_cards[8][15] と *field_status の情報を元に提出するカードを選択し、結果を out_cards に格納する」**ことを意味します。

初期状態での select_submit_cards は、次のようなものです。

Listing 2: 関数 select_submit_cards

```
1  void select_submit_cards(int out_cards[8][15],int my_cards[8][15],
2                           state *field_status)
3  {
4    int select_cards[8][15];
5
6    clear_table(select_cards);
7
8    if(field_status->is_rev==0){
9      // 革命中ではない
10     if(field_status->is_no_card==1){
11       // 場にカードが無いとき
12       select_cards_free(select_cards, my_cards, field_status);
13     }else{
14       // 場にカードが有るとき
```

```
15          select_cards_restrict(select_cards,my_cards, field_status);
16        }
17      }else{
18        // 革命中
19        if(field_status->is_no_card==1){
20          // 場にカードが無いとき
21          select_cards_free_rev(select_cards, my_cards, field_status);
22        }else{
23          // 場にカードが有るとき
24          select_cards_restrict_rev(select_cards, my_cards, field_status);
25        }
26      }
27      copy_table(out_cards, select_cards);
28    }
```

革命中であるか否かで場合分けを行い、その後、場にカードがあるか否かで場合分けを行っています。select_cards_free は自分の手持ちのカードの中で一番弱いカードを単騎で提出し、また、select_cards_restrict は場が単騎の時に自分の手持ちのカードのうちで一番弱いカードを単騎で提出します。select_cards_free_rev と select_cards_restrict_rev は、初期の状態ではパスをします。フローチャートとして書くと、図 1 のようになります。

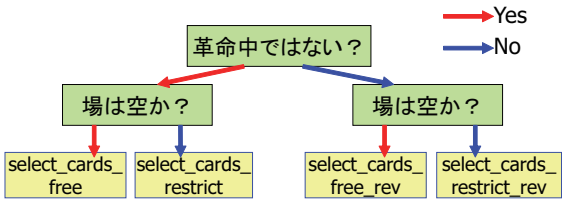


図 1: select_submit_cards のフローチャート

select_cards_restrict のフローチャートを図 2 に示します。この関数のプロトタイプ宣言は以下の通りです。

Listing 3: 関数 select_cards_restrict のプロトタイプ宣言

```
1  void select_cards_restrict(int select_cards[8][15],int my_cards[8][15],
2                             state *field_status);
```

この関数は、自分の手札情報の配列 my_cards と場の情報 field_status から、場が単騎の場合のみ、出せるカードのうちで一番弱いカードの情報を配列 select_cards に乗せます（つまり、どこか一カ所だけに 1 が入り、あとはすべて 0 となります)。場が縛られているときには、

1. 場のスート以外のカードを、関数 remove_suit を用いて、手札情報から削除
2. 場のカードよりも弱いカードを、関数 remove_low_card を用いて、手札情報から 削除
3. 残ったカードの内、一番弱いカードを 1 枚、関数 search_low_card を用いて探す

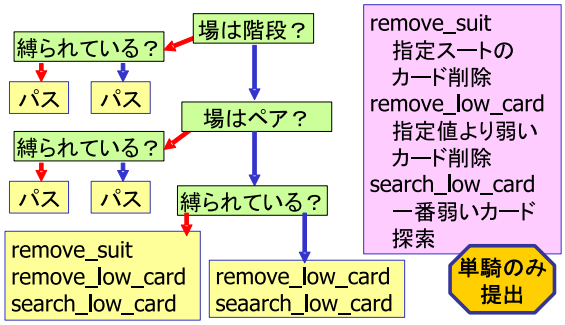


図 2: select_cards_restrict のフローチャート

ことでカード選択を行っています。