プログラミング演習
～総合演習 2～

1　目的

自分の”考え”を”プログラム”として実装する．また，ファイルを分割することでエラー (バグ) を減らす.

1.1　ファイルの分割

総合演習 1 にて「自分の”考え”を”プログラム”として実装」した．その実装方法は一つのファイル”player.c”を作成し，その中で「スコア計算法」と「腕の選択法」に関するプログラムを作成するものであった．しかし，一つのファイルの一つの (main) 関数の中に複数の「やること」を書くと，エラー (バグ) が生じた時に発見しにくく影響が大きい．例えば，腕の選択法のバグがスコア計算法に影響し，不可能なスコアが出る可能性も高い.

そこで，一つのファイルには一つの「やること」を書き，分割することで悪い影響を防ぐ．ここでは，「スコア計算法」と「腕の選択法」に注目し，二種類のファイルに分割する (ヘッダがあるので 3 つのファイルに分割).

- collect.c
  - スコア計算法．また main 関数．みんなに共通なプログラムとして使用．変更不可.
- player.h, player.c
  - 腕の選択法．ここに「自分の考えた腕の選択法」を実装.

2　製作対象 バンディットプレイヤープログラム

ここでは，総合演習 1 で考えた bandit00 用のプレイヤープログラムを分割する．collect.c(2.2), player.h(2.3)，player.c(2.4) をひな形に，player.c のみを書き換えて総合演習 1 と同等のものを作成する.

2.1　準備

ディレクトリ ”programming14”を作成する．今回の演習では，プログラムの作成や必要ファイルのダウンロードは，programming14 ディレクトリで行う.

2.2 collect.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "bandit.h"
4  #include "player.h"
5
6  #define MAX_TRIAL 100000
7
8  int main(){
9    /* 変数定義・初期化 */
10   int i,j,select_arm=0;
11   double reward=0.0,score[10000], max_score=0.0, tmp_score;
12   for(i=0 ; i<10000 ; i++){
13     score[i]=0.0;
14   }
15
16   init_bandit();              /* バンディット初期化 */
17   init_player();              /* プレーヤー初期化 */
18   set_arm_num(get_arm_num()); /* バンディットの腕の数を取得 */
19
20   /* MAX_TRIAL 回まで自動実行 */
21   /* 連続した 10000 回のうち最大のスコアを自動計算・更新 */
22   for(i=0 ; i<MAX_TRIAL ; i++){
23
24     /* 意思決定・それによるバンディットの実行*/
25     select_arm = decision_making(reward);
26     reward = bandit(select_arm);
27     if(reward < 0.0) reward = 0.0;
28
29     /* 連続した 10000 回の最大スコアの確認 */
30     tmp_score=0.0;
31     for(j=0 ; j<10000 ; j++) tmp_score += score[j];
32     if(tmp_score > max_score) max_score = tmp_score;
33
34     /* 連続した 10000 回のスコアを更新            */
35     /* score[0] 〜score[9999] に対し,          */
36     /*   最も古いもの score[9999] を消し,       */
37     /*   一個ずつずらし (score[j] = score[j-1]) */
```

```
38        /*   最も新しいものを score[0] に入れる       */
39        for(j=9999 ; j> 0 ; j--) score[j] = score[j-1];
40        score[0] = reward;
41      }
42
43      printf("最大総獲得報酬: %lf\n", max_score);
44      close_player();
45      return 0;
46    }
```

2.3 player.h

```
1   void init_player();
2   void close_player();
3   void set_arm_num(int arm_num);
4   int decision_making(double previous_reward);
```

2.4 player.c

```
1   #include "player.h"
2
3   static int _arm_num=0; /* このファイルないでしか見えないグローバル変数 */
4
5   void init_player(){
6
7     return;
8   }
9
10  void close_player(){
11
12    return;
13  }
14
15  void set_arm_num(int arm_num){
16    if(arm_num>0){
17      _arm_num = arm_num; /* 使い方の例 */
18    }
19    return;
20  }
```

```
21
22  int decision_making(double previous_reward){
23    /* _arm_num を使えます */
24    /* ここに、bandit00〜bandit08 を”解く”プログラムを書く */
25    /* 例:3 本腕のバンディットなら 1 番目の腕から順に選択する */
26    static int my_select = 0;
27    my_select++;
28    if(my_select > _arm_num) my_select = 1;
29
30    return my_select;
31  }
```

2.4.1 注意

以下のことに注意してプログラムの作成を行うこと．

- player.c の中で#include ”bandit.h”を使用してはいけない
  - 総合演習 1 と同様に，自分で作成するプログラム中に，バンディット関数・bandit() の関数を使用してはいけない．
- srand 関数を使用してはいけない
  - 総合演習 1 と同様である．srand 関数はプログラムの中で「一回だけ実行」するように出来ている．srand 関数は init_bandit の中で実行しているので，再び使用するとプログラムが期待通り動かなくなる可能性がある．

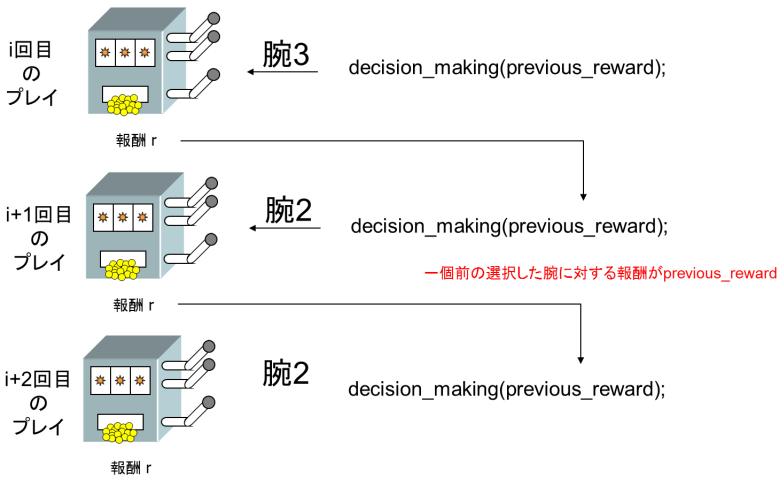

図 1　現在の腕，前回の腕と報酬

2.4.2 指針

総合演習 1 との関連性を考えながら 2.2 collect.c を見ると，重点を置いて考えていた select_arm の決定方法は，decision_making() 関数で行われている (collect.c の 25 行目)．この decision_making() 関数は，2.4 player.c の 22 行目からの関数である．総合演習 2 では，decision_making() 関数を中心に作っていくことで，select_arm の選択法をつくることとなる．

ここでは，ある回にどの腕を選ぶかを考える際に，前回選んだ腕はいくらの報酬であったのか，という情報を基に選択する．前回選んだ腕は，前回 decision_making() 関数で return した数であり，それに対する報酬が previous_reward で与えられるので，これらの情報を基に今回どの腕を選ぶか考えることとなる (図 1)．なお，一回前の腕と報酬だけでなく，過去全ての腕と報酬を用いる場合には，各回ごとに保存していけばよい．

2.4.3 player.c 概要

作成対象は，player.c(プレーヤー:N 本腕バンディットを解く人) である．collect.c から使える player.c の関数は以下のようになる．

- void init_player();
  - 変数やポインタ，動的配列など，なんか初期化したければここで初期化する．
- void set_arm_num(int arm_num);
  - バンディットの腕の数 (n) を知るための関数．collect.c から渡される数を_arm_num に入れるだけなので，変更しなくてよい．
- int decision_making(double previous_reward);
  - decision_making とは，意思決定の意味である．つまり，バンディットのどの腕を選択するかを考える関数である．この関数の返り値は，選択する腕である．引数は，意思決定 (次にどの腕を選ぶか) を考え行うために必要と思われる情報とした．引数 previous_reward は前回選択した腕でいくら報酬がもらえたか，である．試行の第 1 回目は”前回”がないので，この場合には”0.0”が与えられる．
- void close_player();
  - なんか終了処理をする. 特にポインタなどを使っている場合、この中で free をするのに使う．

2.4.4 Tips:ファイルスコープ変数と static 変数

ファイルを分割したことで，従来使えたはずの変数が使えなくなることがある．それらに変わる変数として二種類の変数を紹介する．

まずファイルスコープ変数は，ファイル内の関数ならどこからでも使える変数である．2.4 player.c では，3 行目の

- static int _arm_num=0;

が該当する．この変数は，プレイヤーがプレイするバンディットが何本腕なのかを記憶する変数である．最初に使われるのは，main 関数のある collect.c の 17 行目，player.c では 15 行目の set_arm_num 関数である．collect.c の 17 行目で，get_arm_num() よりバンディットの腕が set_arm_num() に渡される．player.c 中 17 行目で

- _arm_num = arm_num;

とし，_arm_num にバンディットの全体の腕の数が代入され保存される．ここで保存された腕の数は，player.c 中であれば他の関数からも扱える．例えば，player.c 中の 28 行目で，if 文の中で用いられている．

次に，static 変数について説明する．decision_making() 関数では，前回選んだ腕とそれに対応する報酬 (previous_reward) から腕の選択を考える．この際，選択する腕を

- int my_select = ...

と普通の変数として保存すると，decision_making() 関数を処理し終えたらなくなってしまう．このように，

- ある関数内でしか使わない．
- その関数が終わっても変数の情報は保持しておき，次回その関数が呼び出された時に前回の情報をそのまま使いたい．

という場合，static 変数とするとよい．static 変数 (player.c の my_select) の場合，初期化 (player.c の 25 行目) は最初の一度だけ実行され，二回目以降 (関数が 2 度目，3 度目と呼ばれた場合)，初期化はされず前回保持した値がそのまま残っている．例の場合，26 行目，27 行目で変更された値が二回目以降そのまま残っている．

2.5 コンパイル

collect.c, player.h, player.c を作成し，player.c の必要箇所を自分で実装する．色々なバージョンの player.c を作成する場合，player00.c などと名前を変えても良い．

下記の例では，ファイル名 player00.c として保存してコンパイルを行う場合のコマンドである．コンパイル後の名前を gameplay00 とする．

> gcc -o gameplay00 collect.c player00.c bandit00.o

2.6 動作実験

作成したプログラムを実行させ，スコアを出力させる．またより良いスコアを出せるようにプレイヤープログラムを改良する．

2.7 やってみよう

bandit01 に対してもやってみよう