アルゴリズムとデータ構造II Ex12Q2

擬似乱数生成のライブラリ化について

目的

授業で作った中で一番自然な擬似乱数生成アルゴリズムである`next_rnd3()`を用いた自前の擬似乱数生成ライブラリを作ること。

やるべきこと

`next_rnd3()`および、そのために必要な関数や変数などを別ソースファイルに記述する。

効果

擬似乱数生成関数をこれから作成するプログラムで利用するために、毎回プログラムのソース内に擬似乱数生成関数やそれに関わる変数などを全てコピー&ペーストすることは、本質的なプログラムの構造を不明確にし、バグの原因に繋がりやすくなる。これらのリスクをライブラリ化することで回避する。

また、生成される乱数列が実行タイミングによって変化するため、やや現実的な乱数に見える。

ライブラリ化の手順

1. `rnd3.c`をコピーして、`myrand.c`を作る。 `cp rnd3.c myrand.c`
2. 乱数列のseedを乱数生成ごとに変更できるように修正をする。
    1. 以下の関数`void my_srand()`を`myrand.c`に書く。

```
void my_srand(void){
  time_t stamp;
  struct tm *t;

  stamp = (int)time(NULL);
  t = localtime(&stamp);

  x2 = t->tm_sec; // seed

  // add init_rnd()

}
```

`my_srand()`内にある変数`x2`が**現在の時間に応じたseed**になる。

2. `myrand.c`に`time.h`のinclude分を追加する。

   `#include <time.h>`

3. (`rnd3.c`をコピーしたので、`main`関数があるため) `mian`関数を消す。

- `#ifdef`などでコンパイルしないようにするのも可

4. `init_rnd()`関数を、`my_srand()`関数の最後に呼び出すようにする。
5. 消した`main`関数で`init_rnd()`を呼んでいたため、ここで呼ぶことで対応する。
6. いつまでも`next_rnd3()`では不自然なので、関数名を`my_GetRand()`に変えて、以後、これを「擬似乱数生成のための関数」として扱う。

使用方法

1. 擬似乱数を使用するプログラムのソースファイル内に、次の関数を`extern`宣言する。
   1. `my_srand()`
   2. `my_GetRand()`

```
extern void my_srand();
extern int my_GetRand();
```

2. プログラムの出来るだけ早い段階で、`my_srand()`を1回だけ実行する。
   - seedの初期化のため
3. 以上、乱数が欲しい時に、`my_GetRand()`を実行。
4. コンパイル方法は

   `gcc your_program.c myrand.c`

   と、一緒にコンパイル。