第 12 章

最終目標

時間貸し駐車場（コインパーキング）の自動精算機の動作を模倣（エミュレート）しましょう。モデルとする駐車場は、次のようなものです。

- 駐車場の入り口を通過すると、時刻を印刷したカードが発行されます。
- 駐車場から出るときには、このカードを出口の自動精算機に通します。
- 駐車した時間の長さによって駐車料金が決定され、硬貨や千円札などで支払うと、（バーが上がるなどして）場外に出られます。

時刻は 24 時間制で扱うことにします。簡単のため、24 時を越えて駐車することは考えず、1 日のうちに必ず精算することにします。駐車料金は「10 分まで 100 円」を繰り返し適用します。（つまり 11 分は 200 円です。）支払いに使えるお金は、5000 円，1000 円，500 円，100 円の 4 通りにします。

1 日あたりの料金の上限などはひとまず考えませんが、自由課題を設けるので、自由に条件を考えてみてください。駐車場によっては、出入り口にバーがなく、駐車位置のセンサーと車止めで管理しているタイプもあります。そのエミュレートをするのもよいでしょう。

コラム： 単体テスト・結合テスト

関数を一つ作るごとに、その関数の単独の動作テストをしましょう。この作業は**単体テスト** (ユニットテスト・unit test) と呼ばれます。関数の引数（入力データ）を自由に操作できるので、境界値分析（☞**??**ページのコラム）も行いやすいです。

単独の動作が正しいとなれば、次はプログラムに組み込んでの**結合テスト**（統合テスト・integration test) です。結合テストは、プログラムが大きくなって、テストそのものもにも手間がかかりますし、動作不良があっても原因を探るのが難しくなっています。単体テストをちゃんとやっておかないと、後で苦労します。

12.1　規定課題

規定課題で作るプログラムの動作は、以下のように決めておきます。

1. 駐車開始時刻を尋ねます。カードの読み取りに相当する動作です。（利用者は時，分，秒を入力します。）
2. 現在時刻を取得して、経過時間を元に駐車料金 (fee) を決定します。
3. 残額（最初は fee）を表示して、お札や硬貨を 1 枚ずつ受け付けます。（利用者は入金する金額を入力します。）
4. 入金額だけ残額を減らします。まだ残額があれば 3. に戻ります。
5. fee を越えて入金された金額は、お釣り (change) として返却します。「x 円硬貨 y 枚」のように表示します。

実行例を載せておきますので、動作を頭に入れておいてください。赤の斜体の文字は、キーボードから入力した文字です。

ソースコード 12.1 の実行例

```
駐車開始時刻を入力してください
時 => 10
分 => 0
秒 => 0
現在時刻は 10:53:31 です
経過時間は 3211 秒です。
料金は 600 円です。

残額は 600 円です。
お金を投入してください(5000, 1000, 500, 100円のみ) => 10
お金を投入してください(5000, 1000, 500, 100円のみ) => 100
残額は 500 円です。
お金を投入してください(5000, 1000, 500, 100円のみ) => 100
残額は 400 円です。
お金を投入してください(5000, 1000, 500, 100円のみ) => 1000
お釣りは 600 円です。

500 円硬貨 1 枚
100 円硬貨 1 枚
```

12.1.1　部品の作成

各章の学習が終わった時点で、以下で指示する部品となる関数を作ってください。これらの関数は、最後に結合してプログラムを完成させるので、今の段階では単独での動作を確実にしておきましょう。

3 章

- `int sec2fee(int sec)`
  sec 秒駐車したときの駐車料金を返します。料金は 10 分までごとに 100 円とします。0 秒は 0 円です。マイナス秒は考えなくて構いません。

10 分（600 秒）の割り算を**切り上げ**で行なう必要があります。しかし、C 言語の整数の割り算は切り捨てです。double で計算して int にキャストしても、小数点以下は切り捨てられます[a]。

整数に切り上げる a/n の慣用句は「`(a+n-1)/n`」です。n-1 の意味は、**??**章の練習問題**??**. を思い出して考えてください。

[a] 数学関数の `ceil()` は（double のまま）整数値へ切り上げます。しかし、さらに int への型変換も必要ですから、かなり遠回りになります。

6 章

- `int is_coin_note(int value)`
  value が受け取り可能なお札や硬貨の金額であれば論理型の `TRUE` を、そうでなければ `FALSE` を返します。

value が 5000, 1000, 500, 100 のいずれかの場合に `TRUE` を返します。

- `int get_coin_note(void)`
  キーボードから金額を受け取り、受け取り可能なお札や硬貨の金額であれば、その値を返します。そうでなければ、再び金額を受け取り直します。

キーボードから数値を受け取るには、`scanf()` あるいは**??**ページのソースコード**??**の `input_int()` を使えばいいでしょう。受け取り可能な金額かどうかの判定に `is_coin_note()` を呼び出します。**必ず return に到達する**関数になるよう注意してください。

受け取れない金額に「受け取れません」と表示することは、最初の段階では考えなくて構いません。（**??**項のように、例外的な処理が必要になります。）

7 章

- `int sec_diff(int h1, int m1, int s1, int h2, int m2, int s2)`
  h1 時 m1 分 s1 秒から h2 時 m2 分 s2 秒までの、経過秒数を返します。

時分秒のように、60 倍ずつの位取りのある数値を扱うと、

- 加算では繰り上がり
- 減算では繰り下がり

が起こり、処理が複雑になります。
そこで、「h 時 m 分 s 秒」を、「0 時 0 分 0 秒」を起点とする経過秒数に変換し、その秒数で加減算を行う、という手法がよく用いられます[a]。こうすれば、繰り上がりや繰り下がりが起らず、処理が単純になります。
つまり、`sec_diff()` を実現するために、次の関数を補助的に作るとよいでしょう。

- `int hms2sec(int h, int m, int s)`
  0 時 0 分 0 秒を起点とする、h 時 m 分 s 秒までの経過秒数を返します。
- `void print_sec2hms(int sec)`[b]
  0 時 0 分 0 秒から sec 秒経過した時刻を「h 時 m 分 s 秒」の形式で表示します。
  (☞**??**ページの問題**??**.)

[a] UNIX の時刻は、1970 年 1 月 1 日 0 時 0 分 0 秒を起点とする、経過秒数で管理されています。この値を 32 ビット符号付きの型で扱うと、2038 年 1 月 19 日 3 時 14 分 7 秒の次にオーバーフローするというのが「2038 年問題」です。
[b] 本来は h 時 m 分 s 秒に変換した h, m, s の 3 つの値を返したいところですが、関数 1 つで 3 つの値を返す手法をまだ習得していませんので、ここでは即座に表示することにします。

9 章

- `void put_change(int x)`
  x 円のお釣りを返すのに、1000 円札と 500 円と 100 円硬貨を組み合わせて、「a 円 b 枚」の形式で表示します。紙幣と硬貨の合計枚数が最小になる払い方にしてください。

高額のものから枚数を決めていくことを含めて、**??**章の練習問題**??**. とほぼ同じです。今は配列の知識があるので、以前のプログラムを書き換えてみましょう。金額違いで同じ処理を繰り替えしているところを、同じ部分をループ処理に、異なる部分を配列変数にしましょう。こうしておくと、紙幣や硬貨の種類が変わっても、配列を修正するだけですみます。

10 章

- `struct tm *get_current_time(void)`
  現在時刻を返します。実装は以下の通りとします。

```
#include <time.h>
struct tm *get_current_time(void) {
    time_t t = time(NULL);
    return localtime(&t);
}
```

戻り値の `struct tm` は <time.h> で定義されていて、メンバの `tm_hour` に時 (0–23)，`tm_min` に分 (0–59)，`tm_sec` に秒 (0–59) が入ります。(☞**??**節)
6 ページのソースコード 12.1 も参考にして、main() 関数からこれを呼び出し、現在時刻を表示してください。

12.1.2 部品の結合

最後に、これまで作ってきた部品の関数と、次の部品の関数と `main()` 関数を、1 つのファイルに収めて 1 本のプログラムを完成させましょう。

12 章

- `int decide_fee(void)`
  キーボードから対話的に駐車開始時刻を受け取り、現在時刻との差分から経過秒数を求め、駐車料金を決定して返します。
- `int receive_fee(int fee)`
  残額 (fee) を表示し、入金する金額をキーボードから対話的に受け取り、その金額だけ fee を減らします。これを fee が 0 より大きい間繰り返します。fee が 0 以下になったら、お釣りの金額を返します。

3 章で作った `sec2fee()` などを呼び出すと、簡単に駐車料金が決定できます。お金を受け取るためには、6 章で作った `get_coin_note()` を呼び出すと、自動的に金額が限定されます。返すお釣りの値は、正の値になるように気をつけましょう。

```
int main(void) {
    int fee    = decide_fee();      // 駐車料金を決定する
    int change = receive_fee(fee); // 料金を受け取る
    put_change(change);            // お釣りを返す
}
```

全体の概形をソースコード 12.1 に載せておきます。必要なヘッダファイルやプロトタイプ宣言も参考にしてください。

ソースコード 12.1　駐車場の自動精算機（概形）

```
#include <stdio.h>  // BUFSIZ
#include <stdlib.h> // atoi(), exit()
#include <time.h>   // time(), struct tm

#define FALSE 0
#define TRUE  1

/* プロトタイプ宣言 */
int input_int(void);
int sec2fee(int sec);
int is_coin_note(int value);
int get_coin_note(void);
int hms2sec(int h, int m, int s);
int sec_diff(int h1, int m1, int s1, int h2, int m2, int s2);
void put_change(int x);
struct tm *get_current_time(void);
int decide_fee(void);
int receive_fee(int fee);

/* 付録A さらなる成長に向けて */
int input_int(void) {
    char buff[BUFSIZ];
    if (fgets(buff, BUFSIZ, stdin) == NULL) exit(EXIT_FAILURE);
    return atoi(buff); // 文字列→int変換
}

/* 3章 関数（1） */
int sec2fee(int sec) {
    /* ここを作る */
}

/* 6章 繰り返し処理（2） */
int is_coin_note(int value) {
    /* ここを作る */
}

/* 6章 繰り返し処理（2） */
int get_coin_note(void) {
    /* ここを作る */
}

/* 7章 関数（2） */
int hms2sec(int h, int m, int s) {
    /* ここを作る */
}

```

```c
/* 7章 関数（2） */
int sec_diff(int h1, int m1, int s1, int h2, int m2, int s2) {
    /* ここを作る */
}

/* 9章 文字列とポインタ */
void put_change(int x) {
    printf("お釣りは %d 円です。\n\n", x);
    /* ここを作る */
}

/* 10章 構造体 */
struct tm *get_current_time(void) {
    time_t t = time(NULL);
    return localtime(&t);
}

/* 12章 最終目標 */
int decide_fee(void) {
    printf("駐車開始時刻を入力してください\n");
    printf("時 => ");  int h = input_int();  // 付録A
    printf("分 => ");  int m = input_int();
    printf("秒 => ");  int s = input_int();

    struct tm now = *get_current_time();  // 10章
    printf("現在時刻は %2d:%02d:%02d です\n",
           now.tm_hour, /* ここを作る */ 0, 0);

    int sec = /* ここを作る */ 0;
    printf("経過時間は %d 秒です。\n", sec);

    int fee = /* ここを作る */ 0;
    printf("駐車料金は %d 円です。\n\n", fee);
    return fee;
}

/* 12章 最終目標 */
int receive_fee(int fee) {
    /* ここを作る */
}

int main(void) {
    int fee    = decide_fee();     // 駐車料金を決定する
    int change = receive_fee(fee); // 料金を受け取る
    put_change(change);            // お釣りを返す
}
```

12.2　自由課題

自分で条件を自由に設定し、機能拡張してみてください。以下に、機能拡張のヒントを挙げておきます。

- 扱うお札や硬貨の種類を変更する
- `is_coin_note()` や `get_coin_note()`, `put_change()` の金額の種類を、内部で統合する（グローバル変数の配列を作る）
- 現在時刻をコマンドライン引数で指定できるようにする（デバッグに有用）
- 駐車料金を工夫する
  - 単価を変更する（マクロで定義するとよい）
  - 駐車開始直後に無料の時間帯を設ける
  - 1 日あたりの上限金額を設ける
  - 時間帯によって（夜間料金のように）単価を変更する
  - 日付をまたぐことを考慮する[*1]
- 駐車場所ごとに入庫時刻を管理し、出庫時には駐車場所の番号を指定する
  - 駐車場所ごとの入庫時刻をファイルから読み込む
- クーポンなどによる割引

コラム： 実装の意図

時刻を秒単位で管理したのは、過剰スペックかもしれませんが、システムの時刻も秒単位ですから、（自作の）sec_diff() を（標準ライブラリの）difftime() で置き換えるのに好都合です。

put_change() という関数名を print_change() にしなかったのは、単に画面に表示しているのではなく、機器に払い出すよう指示している気分を出すためです。

[*1] `struct tm` で表した時刻を `mktime()` で `time_t` 型の通算秒数に戻し、経過秒数は `difftime()` で求めるとよいでしょう。いずれも `<time.h>` （☞**??**節）の関数です。

索引