Triany Technical Document

http://www.hyuki.com/codeiq/

March 11th, 2013

目次

注意： 本文書はトリアニー問題についての《技術ドキュメント》です。

挑戦者は、本文書に書かれている TLLI (Triany Low Level Interface) と DICTI (Dictionary Interface) をインタフェースに持つモジュールをプログラミングしてください。DICTI は TLLI を用いて実装します。

本文書では、便宜上 C 言語を使ってインタフェースを表記していますが、C 言語で解答しなければならないわけではありません。たとえば、関数をメソッドに置き換えるなど、使用するプログラミング言語で自然な形に置き換えてもかまいません。ただし、その場合でも、対応付けが容易にわかるようなメソッド名をつけてください。

1 Triany Memory

Triany Memory（トリアニー・メモリ）は新素材を用いた二次記憶装置である。

Triany Memory では《非負整数 3 個》を 1 組として管理する。この《非負整数 3 個》のことを **Triany**（トリアニー）と呼ぶ。

> **注意**： 本文書での非負整数とは、プラットホームで取り扱う整数のうち、非負のものを指すと考えてください。

1 個の Triany Memory は、非負整数の個数 − 1 組の Triany を管理する。

個々の Triany には非負整数の管理番号が割り当てられる。この管理番号を **id**（アイディー）と呼ぶ。

0 の id が割り当てられた Triany は存在しない。

1 個の Triany が保持する 3 個の非負整数をそれぞれ、**a, b, c フィールド**と呼ぶ。

図 1 に、Triany の構造を図示する。

図 1: Triany の構造

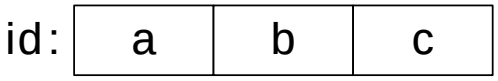
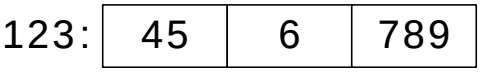


図 2 に、id が 123 で、a, b, c の値がそれぞれ $45, 6, 789$ である Triany を例示する。

図 2: Triany の例



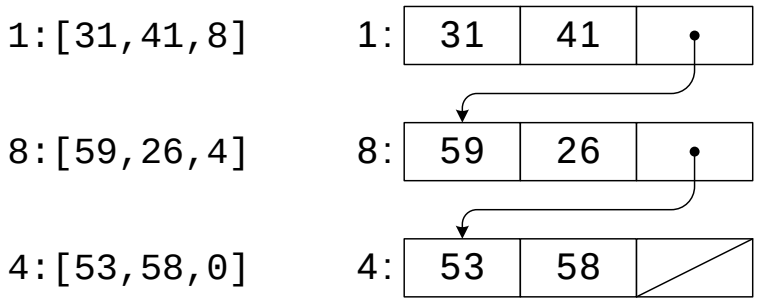
上記の Triany をテキストで表現するときには、図 3 のように表記する。

図 3: Triany の表記

123:[45,6,789]

Triany が保持できる 3 個の非負整数は、id が取り得る範囲と一致するため、リンクを作ることができる。図 4 は、3 組の Triany を用いて 6 個の非負整数を保持するリスト構造を実現した例である。id が 0 に等しい Triany は存在しないため、リストの終端に 0 を用いていることに注意せよ。Triany を用いて、リスト構造以外のデータ構造を作ることも可能である。

図 4: Triany でリスト構造を実現した例



Triany Memory 中の Triany を使うためには、関数 `allocate_triany`（後述）を使って Triany を確保する必要がある。

ただし、Triany Memory 中にはたった 1 個だけ、`allocate_triany` を呼び出さなくても使える Triany が存在する。これを **Root Triany**（ルート・トリアニー）と呼ぶ。Root Triany の id は 1 に等しい。

0 でもなく、Root Triany の id（1）でもなく、関数 `allocate_triany` で確保された Triany の id でもない非負整数を、**不正な id** と呼ぶ。

2 TLLI - Triany Low Level Interface

TLLI(Triany Low Level Interface) は、Triany Memory 中の Triany を管理するための低レベルインタフェースである。以下の 8 個の関数が存在する。

> **注意**： 挑戦者は、TLLI をシミュレートするモジュールを作成してください。

- `int root_triany(void);`
  関数 `root_triany` は、Root Triany の id を得る関数である。戻り値は常に 1 となる。Root Triany の a,b,c の各フィールドの値は、`set_a`,`set_b`, あるいは `set_c` で設定されるまで 0 である。
- `int allocate_triany(void);`
  関数 `allocate_triany` は、新たな Triany を確保し、その Triay の id を得る関数である。戻り値が 0 または 1 になることはない。関数 `allocate_triany` を呼ぶたびに戻り値が増えていくという保証はない。新たな Triany の確保に失敗することはないものとする。得られた Triany の a,b,c の各フィールドは、0 で初期化されている。使わなくなった Triany を解放する必要はない。
- `void set_a(int id, int value);`
  関数 `set_a` は、id で指定された Triany の a フィールドに value の値を設定する。設定は何度でも行うことができ、最後に設定した値が有効になる。id が不正な id または 0 の場合には何も起きない。value が負の場合には何も起きない。
- `void set_b(int id, int value);`
  関数 `set_b` は、b フィールドに対して関数 `set_a` と同様の処理を行う。
- `void set_c(int id, int value);`
  関数 `set_c` は、c フィールドに対して関数 `set_a` と同様の処理を行う。
- `int get_a(int id);`

関数 get_a は、id で指定された Triany の a フィールドの値を返す。id が不正な id または 0 の場合には何も起きない。

- `int get_b(int id);`
  関数 get_b は、b フィールドに対して関数 get_a と同様の処理を行う。
- `int get_c(int id);`
  関数 get_c は、c フィールドに対して関数 get_a と同様の処理を行う。

2.1 TLLI Example

TLLI の使用例を以下に示す。
関数 setup_list は、図 4 に示したリスト構造を作るコード例である。

```
1  void setup_list(void)
2  {
3      int x = root_triany();
4      int y = allocate_triany(); // y の値を仮に 8 とする
5      int z = allocate_triany(); // z の値を仮に 4 とする
6
7      set_a(x, 31);   // 1:[31,41,8]
8      set_b(x, 41);
9      set_c(x, y);
10
11     set_a(y, 59);   // 8:[59,26,4]
12     set_b(y, 26);
13     set_c(y, z);
14
15     set_a(z, 53);   // 4:[53,58,0]
16     set_b(z, 58);
17     set_c(z, 0);
18 }
```

関数 print_list は、図 4 に示したリスト構造に保持されている数値を表示するコード例である。

```
1   void print_list(void)
2   {
3       int t = root_triany();
4       while (t != 0) {
5           int a = get_a(t);
6           int b = get_b(t);
7
8           printf("%d,", a);
9           printf("%d,", b);
10
11          t = get_c(t);
12      }
13      printf("\n");
14  }
```

実行結果は、たとえば 31,41,59,26,53,58 となる。

3 DICTI - Dictionary Interface

DICTI(Dictionary Interface) は、1 以上の整数についての「辞書」を作るインタフェースである。「**辞書**」とは、「キー」と「値」の対（**エントリ**）を多数管理するデータ構造で、一つの辞書の中に、同じキーを持つエントリは複数個存在しないものを指す。DICTI では、キーと値に 1 以上の整数を使うことができる。

> **注意**： 挑戦者は、TLLI を用いて DICTI を実装してください。また、DICTI を呼び出す《辞書プログラム》を作成してください。《辞書プログラム》は《テストデータ》を使って DICT を呼び出し、find_entry の戻り値の合計を解答の 1 行目に記載します。《テストデータ》はファイル testdata.txt として与えられています。

- `void set_entry(int key, int value);`
  関数 set_entry は、key と value を対とするエントリを辞書に挿入する。key と value はいずれも 1 以上の整数とする。key と value のいずれかが 0 以下の場合には何も起きない。与えられた key をキーに持つエントリがすでに存在するなら、そのエントリの値は value で上書きされる。すなわち、辞書の中で、ある key に等しいキーを持つエントリはたかだか 1 個である。
- `int find_entry(int key);`
  関数 find_entry は、key をキーに持つエントリを検索し、もしも辞書中に存在したら、そのキーに対応する value を返す。key をキーに持つエントリが辞書中に存在しなかったら、0 を返す。

3.1 DICTI Example

DICTI の使用例を以下に示す。

関数 sample_dicti は、いくつかのエントリを挿入・検索する。

```
1   void sample_dicti(void)
2   {
3       int n;
4       int sum = 0;
5
6       set_entry(123, 1);
7       set_entry(456, 2);
8       set_entry(789, 3);
9
10      n = find_entry(123);
11      printf("%d\n", n); // => 1
12      sum += n;
13
14      n = find_entry(999);
15      printf("%d\n", n); // => 0
16      sum += n;
17
18      n = find_entry(789);
19      printf("%d\n", n); // => 3
20      sum += n;
21
22      set_entry(456, 90);
23      n = find_entry(456);
24      printf("%d\n", n); // => 90
25      sum += n;
26
27      set_entry(456, 6);
28      n = find_entry(456);
29      printf("%d\n", n); // => 6
30      sum += n;
31
32      printf("%d\n", sum); // => 100
33  }
```

索引

本文書に登場する Triany Memory は架空のデバイスである。