『プログラム言語論』 Short Quiz (2012/05/14)

Short Quiz 1. 「ヒープの役割はなにか？」**解答例**

解答: ヒープは、プログラム言語処理系が実行時に使用するメモリのうち、スタック (やレジスタ) 以外の部分であり、プログラム実行時に生成されるデータ (「対」や「文字列」などのデータ構造、関数クロージャ、オブジェクト指向言語におけるオブジェクトなど) を格納するために用いる。

解説: C, ML, Lisp 言語など、ブロック構造を持つ言語の実行時には、ブロック呼びだしに対応する「スタックフレーム」とよばれるデータのかたまりを、スタックに格納する方式を取る。(関数など、ブロックを 1 つ呼びだすと、スタックフレームが 1 つ、スタックに積まれる。) スタックフレームには、そのブロックに局所的な変数とその値 (およびその他のデータ) を格納する。

しかし、これらの言語でも、「対 (ペア)」、「関数クロージャ」、「オブジェクト」など、構造を持ったデータの lifetime (生存期間、そのデータが生成されてから、「ごみ」になるまでの間) は、ブロックの生存時間に対応しない。その理由を考えるために、「関数 f で、対のデータを生成し、そのまま、対を関数の返り値とする」という場合を考える。

```
OCaml:
 let pair =
   (let f x = (x, x+1) in (f 10))
 in
   fst pair


Scheme:
  (let ((pair
          (let ((f (lambda (x) (list x (+ x 1)))))
            (f 10))))
   (car pair))
```

この場合、関数 f の呼びだし (f 10) が終わった後も、その結果である (10,11) という対を pair という変数にいれて利用する。つまり、対のデータは、(f 10) の呼び出しがおわったあとも (f 10 に対応するスタックフレームが除去された後も) 「生存」している必要がある。

関数クロージャやオブジェクトも同様であり、「関数 f の中で、関数クロージャやオブジェクトを生成し、それを関数 f の返り値とする」場合などは、それらのデータを、関数 f に対応するスタックフレームに格納してはいけない。よって、ヒープに格納する。

ヒープに格納するデータは、スタックと違って「いつまで有効か」がはっきりせず (データごとに異なる)、「有効でなくなったら自動的に捨てられる」ということがないので、言語処理系としては、ヒープ中の無効になったデータを回収する「ごみ集め」を時々起動して、ヒープを有効に使うようにする必要がある。(ただし、C 言語では、ごみ集めは、プログラマが自分で記述しなければいけない。)

Short Quiz 2. C 言語で、「malloc で獲得したヒープ領域を、free し忘れる」ことを防ぐ、良い方法はないか？

解答の前に「free し忘れを防ぐ、完璧で自動的で、かつ、実行性能のよい方法」はなさそうだ。なぜなら、完璧にそんなことが自動的にできて、かつ、処理性能を落とさない、ということが可能なら、C 言語処理系が当然組みこんでいるはずだから。

したがって、完璧なすごい解答はあり得ない。この問題は、「不完全でよいので、どんなアイディアがあり得るか」を自由に書いてもらう、という意図である。

1 つだけ、多少まともかもしれない方法. 1 つは、malloc で獲得したヒープ領域に、データを書きこむとき、そのデータを何か所から参照しているかをメモしておく、という方法がある。たとえば、関数クロージャを生成した

が、それを使う場所が 1 か所だけであり、その 1 か所の使用が終了したら、その関数クロージャは「ごみ」である。このように、「自分が何か所から使われるか」をデータの側に書きこんだものを、reference count という。

(なお、C 言語の場合は、ポインタは単なるデータなので、それを別の変数にコピーすることも簡単にできてしまうので、reference count を正確に記録しつづけるのは、実装はかなり難しい。また、完璧にやろうとしたら、ポインタ演算がとても遅くなってしまう。これは C 言語の場合は許しがたいだろう。)