問題 I

(a) 以下の Prolog のプログラムの空欄を埋めて掛け算のプログラムを完成させよ．なお，mult(X,Y,Z) は，X とY の積がZ であることを表すようにしたいものとする．

```
add(0, X, X).                              /*** 足し算の定義 ***/
add(s(X), Y, s(Z)) :- add(X, Y, Z).

mult(0, X, [ 0 ]).                  /*** 掛け算の定義 ***/
mult(s(X), Y, Z) :- add(Y, W, Z), mult(X, Y, [ W ]).
```

(b) 次の各クエリーに対する結果を予想せよ．

1. `?- mult(s(s(0)), s(s(s(s(0)))), X).`

   `X = s(s(s(s(s(s(s(s(0))))))))`

2. `?- mult(s(s(0)), X, s(s(s(s(s(s(0)))))).`

   `X = s(s(s(0)))`

3. `?- mult(X, s(s(0)), s(s(s(s(0))))).`

   `X = s(s(0))`

4. `?- mult(X, s(s(0)), s(s(s(0)))).`

   `No.`

(c) （発展問題）実際に Prolog の処理系で上のプログラムを実行して (b) の予想を確かめよ．
Prolog の処理系としては， SWI-Prolog(http://www.swi-prolog.org),
GNU Prolog(http://pauillac.inria.fr/~diaz/gnu-prolog/) などがあるので，自分の PC にインストールすること．

解説 上の入力例に対しては意図した通りに動くはず．ただし、上の定義ではmult(X,Y, s(s(0))) のような問い合わせに対してはうまく計算してくれない．これは， Prolog の証明探索の戦略によるものであり，理由を知りたい人は，論理プログラミングの本で勉強してほしい．参考までに，以下にmult(X,Y, s(s(0))) のような問い合わせに対しても計算できるプログラムを示す（講義中の Prolog のデモではこの定義を用いた）．

```
mult(0, X, 0).
mult(s(X), 0, 0).
mult(s(X),s(Y), Z) :- add(s(Y), W, Z), mult(X, s(Y), W).
```

問題 II

次の２つのプログラムを並行に実行させ、 producer 側で生成された各データが順に consumer で消費されるようにしたい。下の問 (a), (b) に答えよ.

- producer 側のプログラム：

```
while true do
 (datap := produce_data();   /*** データを生成 ***/
  while counter = n do skip; /*** buffer に空きができるまで待つ ***/
  buffer[in] := datap;        /*** データを buffer に書き込む ***/
  in := (in+1) mod n;          /*** 次に書き込むべきインデックスを計算 ***/
  counter := counter+1       /*** データの個数を更新 ***/
 )
```

- consumer 側のプログラム:

```
while true do
 (while counter = 0 do skip;  /*** buffer にデータが書き込まれるまで待つ ***/
  datac := buffer[out];      /*** buffer からデータを読み込む ***/
  out := (out+1) mod n;        /*** 次に読み込むべきインデックスを計算 ***/
  counter := counter-1;      /*** データの個数を更新 ***/
  consume_data(datac)        /*** データを消費 ***/
 )
```

ただし、各変数や手続きの意味は以下のとおり。

- `buffer`··· 生成されたデータを格納しておくためのサイズ n の配列。 producer と consumer で共有。
- `counter`··· counter は buffer に格納されている（まだ消費されていない）データの個数を表す。 producer と consumer で共有。
- `in`··· 次にデータを書き込むべき buffer の場所を表す変数。
- `out`··· 次にデータを読み込むべき buffer の場所を表す変数。
- `produce_data`··· 新しいデータを生成するための関数
- `consume_data`··· データを消費する関数

(a) 上の producer と consumer を並行に実行した場合の問題点を指摘せよ。

producer と consumer が`counter:=counter+1` と`counter:=counter-1` を同時に実行した場合に，`counter` の値が正しく更新されない可能性がある.

(b) (a) の問題を回避するために、ロックの獲得・解放命令`lock()`, `unlock()` を挿入したい。どこに挿入すればよいか？

`counter:=counter+1` と`counter:=counter-1` の直前に`lock()`、直後に`unlock()` を挿入すればよい.