【サンプルプログラム】

★ ex07-1.c ここでつくっている関数は以下の仕様をみたすものである.

【形式】

```
unsigned int uint_power(unsigned int a, unsigned int n)
```

【機能説明】

a の n 乗を返します. 「桁あふれ」が生じた場合には正しい値を返す保証はありません.

この中で, 次のことに注意してもらいたい.

- 大域変数（呼び出し側）と局所変数に同じ識別子名を持つものが使われているが, それらは異なる変数である. （違う値が入っている場合があるので一目瞭然である.）
- 関数内部で n の値が変化しているが, 呼び出し側に戻ったときに n の値は変化しているのだろうか？また, それはなぜか？
- 仮引数定義に用いられている識別子名と実引数の識別子名は無関係である.
- 実引数には「定数」を用いることができる.
- 関数プロトタイプ宣言を用いないときには「エラー」が発生する. （先頭の方にある
  `int uint_power(int, unsigned int);`
  という行をコメントアウトしてみよう. ）
- 関数内部で使っている `static unsigned int count` は, この関数を呼び出した回数を記憶するものである. （この関数には本来は必要ないが, 「静的変数」の例として使っているだけである.）この変数の初期化はいつ行われているかに注意しよう.
- 局所変数に初期化を行わないとどうなるかを調べてみよう. つまり, 意図的に局所変数の初期化をやらないことにすると, どんな状況になるのだろうか？(これは uint_power 内での i の値の出力結果を見てみてもわかる.)

【課題】

★ exercise-07-1 次の仕様をみたす関数をつくり, 西暦１６０１年から２０００年までの閏年を全て出力するプログラムを書きなさい.

【形式】

```
int is_leapyear(int year)
```

【機能説明】

西暦 year 年が閏年の場合には 1 を返し, 平年の場合には 0 を返します.

★ exercise-07-2 次の仕様をみたす関数をつくり, ２つの正の整数の最大公約数を求めるプログラムを書きなさい.

【形式】

```
unsigned int gcd(unsigned int a, unsigned int b)
```

【機能説明】

a, b がともに 0 でない場合にはそれらの最大公約数を返します. a, b のいずれか一方が 0 の場合には 0 を返します.

★ exercise-07-3 次の仕様をみたす関数をつくり, ２つの正の整数の最小公倍数を求めるプログラムを書きなさい.

【形式】

```
unsigned int lcm(unsigned int a, unsigned int b)
```

【機能説明】

a, b の最小公倍数を返します.

【エラー】

最小公倍数の値が unsigned int の範囲を越える場合には正しい結果となることを保証しません.

【その他】 電子メールで「今日の講義の感想や意見」を送ってください.

★ ex07-1.c の内容

```
/* $Id: ex07-1.c,v 1.7 2004-05-20 13:09:10+09 naito Exp $ */
#include <stdio.h>

unsigned int    uint_power(unsigned int, unsigned int) ;

int main(int argc, char **argv)
{
  unsigned int  m = 3U, n = 5U ;

  printf("(answer)\t%u\n", uint_power(2U,3U)) ;
  printf("(answer)\t%u\n", uint_power(2U,m)) ;
  printf("(main)\t\tn = %u, m = %u\n", n,m) ;
  printf("(answer)\t%u\n", uint_power(n,m)) ;
  printf("(main)\t\tn = %u, m = %u\n", n,m) ;
  return 0 ;
}

unsigned int uint_power(unsigned int a, unsigned int n)
{
  unsigned int  m = 1U, i ;
  static int    count = 0 ;

  count += 1 ;
  printf("(uint_power)\tn = %u, m = %u, a = %u, i = %u, count = %d\n",
         n, m, a, i, count) ;
  if (n == 0) return 1 ;
  for(i=0;i<n;i++) m *= a ;
  return m ;
}
```

● 前回の講義のキーポイント及び補足

- 「アルゴリズム」とは「問題を解くための『計算手順』」である. 計算機の立場では「与えられた問題を解くために, 問題の条件をみたすどんな入力に対しても正しい結果を得るための計算手順」を示したものである.
  - アルゴリズムは, 問題の条件をみたすいかなる入力に対しても正しい結果を得ることができなければならない. （または, 「結果を得ることができない」という意味での「エラー」を変えさなければならない.）
  - アルゴリズムは, 問題の条件をみたすいかなる入力に対しても**有限回**の操作で終了しなければならない. （そのステップ数は入力に応じて変化する可能性がある.）
  - アルゴリズムの中の手順は「決定的」でなければならない. つまり, 手順の中で「ランダム」な要素があってはならない.

    注意１ ここでの「決定的」という言葉は, 計算機科学における「決定的」という言葉とは異なり, 「確率的」という言葉に対比されるものである.

    注意２ 近年は「確率論的アルゴリズム」というものもあり, 必ずしもこの条件に当てはまらないものも多くなっている.

- 「アルゴリズム」が「正しい」こと, すなわち「正しい出力を得ることができる証明」は, 多くの場合「数学的帰納法」を用いて行われる.

  【具体例】 ユークリッドの互除法では「除算を適用する２つの数の最大公約数」がループで不変であることを使ってアルゴリズムの正当性を証明する.

- アルゴリズムの「計算量」とは, 入力に対して出力を得るために必要な計算ステップ数を, 入力データの何らかの量で表現したものであり, アルゴリズムの「複雑さ」をあらわす量である.

  【具体例】 ユークリッドの互除法における除算回数は, 入力データの大きい方の値を a とおくと, $O(\log a)$ 回である. そこにおける１回の除算の計算ステップは $O((\log a)^2)$ ステップであるので, ユークリッドの互除法の全計算ステップは $O((\log a)^3)$ となる.

- アルゴリズムの「計算量」とは入力データが「大きくなったとき」の計算の複雑さをあらわす指標であり, 計算量が少ない（高速なアルゴリズム）であるからと言って, 「小さなデータ」に関して本当に「速い」のかどうかはわからない.

● 前回の課題の解説

★ exercise-06-2 ２つの正の整数の最大公約数を求めるプログラムを「ユークリッドの互除法」を用いて書きなさい. （以下の２つのプログラムはそれぞれ一長一短がある.）


```
/* $Id: exercise-06-2-1.c,v 1.2 2004-05-20 13:15:58+09 naito Exp $ */
/* ユークリッドの互除法によって最大公約数を求める */
#include <stdio.h>
int     a, b, r ;
int main(int argc, char **argv)
{
  a = 40920 ; b = 24140 ;
  printf("gcd(%d,%d) = ", a,b) ;
  while((r = a%b)) {
    a = b ; b = r ;
  }
  printf("%d\n", b) ;
  return 0 ;
}
```

```
/* $Id: exercise-06-2-2.c,v 1.2 2004-05-20 13:16:00+09 naito Exp $ */
/* ユークリッドの互除法によって最大公約数を求める */
#include <stdio.h>
int     a, b, r ;
int main(int argc, char **argv)
{
  a = 40920 ; b = 24140 ;
  while(b) {
    r = a%b ; a = b ; b = r ;
  }
  printf("%d\n", a) ;
  return 0 ;
}
```

- 上では b が 0 の時に「除算エラー」が発生するため, 実際には b が 0 かどうかを判断しておかなければいけない.
- 下は b が 0 であっても答として a を返すが, 除算回数が上と比較して１回増えてしまう.

$a < b$ の場合であっても, これらのプログラムは正しく動作することを理解して欲しい. また, それぞれのプログラムに対する元のアルゴリズムは次の通り.

▼ 上のプログラム

1. $a = bq + r$ (q は a を b で割った商) をみたす r を求める.
2. r が 0 でないならば, a に b を代入し, b に r を代入して 1 に戻る.
   r が 0 ならば b が求める最大公約数となり, アルゴリズムを終了する.

▼ 下のプログラム

1. b が 0 ならば a が求める最大公約数となり, アルゴリズムを終了する.
2. $a = bq + r$ (q は a を b で割った商) をみたす r を求める.
3. a に b を代入し, b に r を代入して 1 に戻る.