【これまでの実習課題の解説】 以下に解説をしている課題は，「学期末レポート課題」に関係している可能性の高いものです．当然ですが，直接は「レポート課題の解答」にはなっていません．また，課題の完全な解答にはなっていないものもあります．

★ exercise-10-4 与えられた unsigned int 型の数値の素因数を全て求めるプログラムを書きなさい．

素因数分解は「順に割っていく」以外の方法は非常に困難である．そのため，以下のようなアルゴリズムを使えばよい．

1. ２の因子を全て取り出す．
2. 残りは奇数因子だけとなっているので，奇数で順に割っていく．調べている奇数が，素因子で割った商を越えるまでそれを繰り返せばよい．

下にあるプログラム（関数）は，n の全ての素因数を配列 a に戻し，戻り値として，素因数の個数を返すものである．

★ exercise-05-5 ２つの unsigned int 型の整数の商と剰余を計算するプログラムを書きなさい．

このプログラムは「除算の筆算」をそのまま書けばよい．１０進の筆算は，各桁の商が 0 から 9 までの可能性があるため，非常にめんどうな計算になる．２進の筆算に置き換えれば，商は 0 または 1 であり，それを判断するためには，被除数と除数の大小関係がわかればよい．

２進の除算の筆算を実現するアルゴリズムの概略は以下の通り．

1. 除数の桁を被除数の桁にそろえる．
2. 以下の手続きを，被除数が除数より小さくなるまで繰りかえす．
   (a) 被除数と除数の大小関係を比較し，その桁の商を求める．
   (b) 除数を一桁繰り下げる．

★ exercise-14-3 標準入力からテキストデータを読み込み，１行ごとに書かれた「空白で区切られた最大１０個の整数」の和を順に表示するプログラムを書きなさい．

- これを実現するには，標準入力から読み込んだテキストデータを「トークン分解」して，各トークンを int 型の数値に変換する関数を作ればよい．これは store_int_array で実現されている．この関数は戻り値として「読み込んだ整数の個数」を返す．
- store_int_array の実現方法は２通りの方法を示してある．
  1. 標準関数 “strtok” を用いる方法．
     – strtok は第２引数に与えた文字列に含まれる文字を区切り文字としてトークン分解を行う．
     – １回目の strtok の呼び出しでは，トークン分解を行う対象の文字列を第１引数に与えると，先頭のトークンへのポインタが戻ってくる．
     – 同じ文字列を対象に「次のトークン」を得るためには，strtok の第１引数に NULL を与えて呼び出しを行う．「それ以上トークンが見つからない」時には strtok は NULL を返す．

     この strtok 関数は，内部では以前のトークン分解対象の文字列を static 変数として保持している．
  2. 自作の関数 “_strtok” を用いる方法．(exercise-12-6)
     – _strtok は第２引数にトークン分解の対象となる文字列，第３引数に区切り文字の集合を示す文字列を与えると，第１引数に最初のトークンを返し，戻り値には見つかったトークンの最後の文字の次の文字を示すポインタを返す．トークンが見つからないときには NULL を返す．
     – その実装方法は，区切り文字に含まれない最初の位置と，それ以後の区切り文字に含まれない prefix の長さを求め，その部分を第１引数に与えた文字列にコピーする．
     – ２回目以後の呼び出しでは _strtok の戻り値のポインタを第２引数に与えればよい．

★ exercise-14-8 標準入力から１行読み込み，そこに書かれた「算術式」を「逆ポーランド記法」に変換した結果を表示するプログラムを書きなさい．ここでは，入力されたものが「算術式」であるかどうかのエラー判定は必要ないものとします．

一見すると非常に難解に見えるが，実は極めて簡単である．まず，以下の「算術式の構文定義」の意味を見ていこう．

【構文定義】

```
<Expression> ::= <Term> | <Expression> <Additive Operator> <Term>
<Term>       ::= <Factor> | <Term> <Multiplicative Operator> <Factor>
<Factor>     ::= <Identifier> | ( <Expresssion> )
<Identifier> ::= <Alpha Numeric Character>
```

【例１】 “1 + 2 * 3”

これは次のように構文定義に一致する．

1. “1 + 2 * 3” を <Expression> の定義と見比べ，+ に着目して分解すると，“1” “+” “2 * 3” がそれぞれ <Expression>, <Additive Operator>, <Term> となる．
2. “1” は <Expression> の定義の <Term> に一致し，<Term> の定義の <Factor> に一致し，<Factor> の定義の <Identifier> に一致する．
3. “2 * 3” は <Term> の定義と見比べ，* に着目して分解すると，“2” “*” “3” がそれぞれ<Term>, <Multiplicative Operator> <Factor> に一致する．
4. “2 は <Term> の定義の <Factor> に一致し，<Factor> の定義の <Identifier> に一致する．
5. “3 は <Factor> の定義の <Identifier> に一致する．

以上によって “1 + 2 * 3” が算術式の定義に一致することがわかった．
また，この定義から逆ポーランド記法への変換は，

```
<Expression> <Additive Operator> <Term>
                    => <Expression> <Term> <Additive Operator>
<Term> <Multiplicative Operator> <Factor>
                    => <Term> <Factor> <Multiplicative Operator>
<Identifier>        => <Identifer>
( <Expresssion> )   => <Expression>
```

という置き換えを順に行えばよい．上の例では，

```
1 + 2 * 3     ⟹ 1 "2 * 3" +
1 "2 * 3" +   ⟹ 1 2 3 * +
```

となる．

【例２】 “(1 + 2) * 3”

これは「例１」とは異る一致方法になる．

1. “(1 + 2) * 3” は “<Expression> <Addtive Operator> <Term> には一致しない. これに一致すると仮定すると, <Term> が “2) * 3” となり, “)” があるため, 以後の <Factor> の定義に一致しなくなる.
   したがって, “(1 + 2) * 3” は “<Term> と考えて処理を続行する.
2. “(1 + 2) * 3” は “<Term> の <Term> <Multiplicative Operator> <Factor> と一致する. ここで, <Term> は “(1 + 2)” である.
3. “(1 + 2) は <Term> の <Factor> に一致し, <Factor> の (< Expression>) に一致する. ここで, <Expression> は “1 + 2” である.

したがって, これを逆ポーランド記法に変換すると,

```
( 1 + 2 ) * 3     ⟹ "( 1 + 2 )" 3 *
"( 1 + 2 )" 3 *   ⟹ 1 2 + 3 *
```

となる.

このように, 構文規則にそって再帰的に逆ポーランド記法に変換することができる.

【レポート課題のための補足】

★ ex15-1 N 個の int 型の配列要素を小さい順に並び替えるプログラム.

配列要素を指定された順に並び替えることを「**ソート**」 (**sort**) と呼ぶ. 要素数 N に対して $O(N^2)$ の計算時間を要するソートのアルゴリズムとしては, 以下の３種類の方法が有名である. (この他にも, より高速な $O(N \log N)$ の計算時間のアルゴリズム (**quick sort**, **heap sort** など) も存在するが, 少々難しい)

☆ **単純選択法** a[i] から a[N-1] の中で最小のものを探し, それと a[i] を交換する.

☆ **単純挿入法** a[0] から a[i-1] が既にソートされていると仮定して, a[i] を適切な位置に挿入する. (トランプのカードを並び替える方法そのものである.)

☆ **単純交換法** a[j] と a[j-1] が正しい順に並んでいなければそれを交換することを繰り返す. (これは, **bubble sort** と呼ばれ, 最も単純で有名なアルゴリズムである.)

【補足】 情報メディア教育センターの Linux System を利用している場合には,

```
#include <strings.h>
```

を行っている場所を, 全て

```
#include <string.h>
#include <strings.h>
```

に変更してください. (Solaris と Linux でヘッダファイルの仕様が少々異っているようです.)

【その他】 電子メールで「半年間の授業の感想や意見」を送ってください.

★ ex15-1 の内容

```
/* $Id: selection_sort.c,v 1.2 2004-07-17 16:19:39+09 naito Exp $ */
/* i->n の中で最小のものを探し, a[i] と a[k] を交換する */
void    selection_sort(int *a, int n)
{
  int   i, j, k, min ;

  for(i=0;i<n;i++) {
    min=a[i] ; k = i ;
    for(j=i+1;j<n;j++) {
      if (min>a[j]) {   /* i+1 から N の中で最小のものを探す */
    min = a[j] ; k = j ;
      }
    }
    swap(a+i, a+k) ;
  }
  return ;
}
```

```
/* $Id: insersion_sort.c,v 1.2 2004-07-17 16:19:21+09 naito Exp $ */
/* a[0]->a[i-1] が既に整列していると仮定して,     *
 * a[i] を挿入する場所を探す           */
void insersion_sort(int *a, int n)
{
  int   i, j, k, c ;

  for(i=1;i<n;i++) {
    j = 0 ; c = a[i] ;
    while((j<i)&&(a[j] <= a[i])) j++ ; /* 挿入位置の決定 */
    /* a[i] を a[j] に挿入 */
    for(k=i;k>j;k--) a[k] = a[k-1] ;
    a[j] = c ;
  }
  return ;
}
```

```
/* $Id: bubble_sort.c,v 1.2 2004-07-17 16:19:13+09 naito Exp $ */
void bubble_sort(int *a, int n)
{
  int   i, j ;

  for(i=0;i<n;i++) {
    for(j=n-1;j>i;j--) {
      if (a[j] < a[j-1]) swap(a+j, a+j-1) ;
    }
  }
  return ;
}
```

```
/* $Id: sort.c,v 1.2 2004-07-17 16:22:27+09 naito Exp $ */
/* Classical Sort */
#include <stdio.h>
#define N   11

void    swap(int *, int *) ;
void    selection_sort(int *, int) ;
void    insersion_sort(int *, int) ;
void    bubble_sort(int *, int) ;

int main(int argc, char **argv)
{
  int   a[N] = {4,5,3,2,1,8,7,6,9,0,6} ;
  int   i ;

  selection_sort(a1, N) ;
/*    insersion_sort(a1, N) ;   */
/*    bubble_sort(a1, N) ;  */
  for(i=0;i<N;i++) printf("%d ", a1[i]) ; printf("\n") ;
  return 0 ;
}
```


★ exercise-10-4 の内容

```
/* $Id: factorization_lib.c,v 1.2 2004-07-17 16:53:23+09 naito Exp $ */
unsigned int factorization(unsigned int n, unsigned int *a)
{
  unsigned int  count = 0, p = 3 ;

  /* ２の巾を得る */
  while((n&1) == 0) {
    a[count++] = 2 ; n >>= 1 ;
  }
  /* 奇数の素因数を得る */
  while(p <= n) {
    while(n%p == 0) {
      a[count++] = p ; n /= p ;
    }
    p += 2 ;
  }
  return count ;
}
```

★ exercise-14-3 の内容

```
/* $Id: exercise14-3-1.c,v 1.2 2004-07-20 15:51:57+09 naito Exp $ */
#include <string.h>
#include <strings.h>
#include <stdlib.h>
#define DELIMITER       " \r\v\t\n"

char    *_strtok(char *, const char *, const char *) ;

/* s にある「整数」を配列 a に格納する．    *
 * 戻り値は読み込んだ整数の個数        */
int store_int_array(const char *s, int *a)
{
  char  *p ;
  char  buf[MAX_CHAR_PAR_LINE] ;
  int   count = 0 ;

  while((p = _strtok(buf, s, DELIMITER)) != NULL) {
    a[count++] = atoi(buf) ;
    s = p ;
  }
  return count ;
}

/* 文字列 s を区切り文字集合 d に関してトークン分解する    *
 * 最初に見つかったトークンを t に返す．          *
 * 戻り値は s の中で，t の次の文字．          *
 * それ以上トークンが見つからない場合は NULL を返す．     */
char *_strtok(char *t, const char *s, const char *d)
{
  char      *p ;
  size_t    len ;

  if (!*s) return NULL ;
  /* 区切り文字に含まれる prefix の長さを得る */
  len = strspn(s, d) ; p = (char *)s+len ;
  /* 区切り文字に含まれない prefix の長さを得る
   * もし，len == 0 ならば prefix が存在しない */
  if ((len = strcspn(p, d)) == 0) return NULL ;
  /* p <--> p+len はトークン */
  strncpy(t, p, len) ; t[len] = '\000' ;
  return p+len ;
}
```



```
/* $Id: exercise14-3-2.c,v 1.2 2004-07-20 15:51:52+09 naito Exp $ */
#include <string.h>
#include <strings.h>
#include <stdlib.h>
#define DELIMITER       " \r\v\t\n"

/* s にある「整数」を配列 a に格納する．    *
 * 戻り値は読み込んだ整数の個数        */
int store_int_array(const char *s, int *a)
{
  char  *p, *q ;
  int   count = 0 ;

  q = (char *)s ;
  while((p = strtok(q, DELIMITER)) != NULL) {
    a[count++] = atoi(p) ;
    q = NULL ;
  }
  return count ;
}
```

```
/* $Id: exercise14-3-main.c,v 1.3 2004-07-20 15:51:47+09 naito Exp $ */
#include <stdio.h>
#include <string.h>
#include <strings.h>
#include <stdlib.h>
#define MAX_CHAR_PAR_LINE   (1024)
#define N           (10)
int store_int_array(const char *, int *) ;
int main(int argc, char **argv)
{
  char  buf[MAX_CHAR_PAR_LINE] ;
  int   a[N], count, sum, i ;
  while(!feof(stdin)) {
    if (fgets(buf, MAX_CHAR_PAR_LINE, stdin) != NULL) {
      /* 1行を読み，「整数」を a に格納する．*/
      /* count は読み込んだ整数の個数          */
      sum = 0 ;
      if ((count = store_int_array(buf, a)) <= N)
    for(i=0;i<count;i++) sum += a[i] ;
    }
    printf("sum = %d\n", sum) ;
    sum = 0 ;
  }
  return 0 ;
}
```


★ exercise-05-5 の内容

```
/* $Id: exercise05-5.c,v 1.2 2004-07-18 10:02:59+09 naito Exp $ */
#include <stdio.h>

void    div(unsigned int, unsigned int, unsigned int *, unsigned int *) ;

int main(int argc, char **argv)
{
  unsigned int  a, b ;
  unsigned int  q, r ;

  a = 1000211 ; b = 11111 ;

  div(a, b, &q, &r) ;
  printf("q = %u\n", q) ;
  printf("r = %u\n", r) ;
  return 0 ;
}

/* a を b で割った商 q, 剰余 r を求める */
void div(unsigned int a, unsigned int b, unsigned int *q, unsigned int *r)
{
  int   n = 0 ;

  *q = 0 ; *r = 0 ;
  /* a と b のビット数が揃うまで b を左シフトする */
  while(a >= (b<<1)) {
    b <<= 1 ; n += 1 ;
  }
  while(n >= 0) {
    /* a >= b ならば商の該当するビットに 1 をたて，
     * 剰余を計算する */
    if (a >= b) {
      *q += (1 << n) ; a -= b ;
    }
    b >>= 1 ; n -= 1 ;
  }
  *r = a ;
  return ;
}
```

★ ハノイの塔修正版

```
/* $Id: hanoi.c,v 1.1 2004-07-21 11:32:41+09 naito Exp $ */
#include <stdio.h>

void    hanoi(int, int, int, int, int) ;

int main(int argc, char **argv)
{
  hanoi(4, 0, 0, 1, 2) ;
  return 0 ;
}

/* ハノイの塔
 * k: 板の枚数
 * m: 最小の板番号
 * p0,...,p2: 柱番号, p0 -> p2 を実行する
 */
void hanoi(int k, int m, int p0, int p1, int p2)
{
  if (k == 1) {
    fprintf(stdout, "%d %d %d\n", m, p0, p2) ;
    return ;
  }
  hanoi(k-1, m, p0, p2, p1) ;
  fprintf(stdout, "%d %d %d\n", m+k-1, p0, p2) ;
  hanoi(k-1, m, p1, p0, p2) ;
  return ;
}
```