**【課題】**

★ exercise-11-1 「エラトステネスのふるい」を用いて 10000 までの素数を全て求めるプログラムを書きなさい. （無理に「関数」として実現する必要はない.）

★ exercise-11-2 $b$ を 2 以上 36 以下の整数としたとき, unsigned int 型の値を $b$ 進表示した結果を表示するプログラムを以下のような関数を構成して書きなさい.

**【形式】**

```
void radix(char s[], unsigned int a, unsigned int b)
```

**【機能説明】**

unsigned int 型の値 a の b 進表示した文字列を s に返す.

★ exercise-11-3 int 型の値を平衡３進表示した結果を表示するプログラムを以下のような関数を構成して書きなさい.

**【形式】**

```
void blanced_three(char s[], int a)
```

**【機能説明】**

int 型の値 a の平衡３進表示した文字列を s に返す.

★ exercise-11-4 （**難**）$b$ を 2 以上 36 以下の整数としたとき, int 型の値を $-b$ 進表示した結果を表示するプログラムを以下のような関数を構成して書きなさい.

**【形式】**

```
void neg_radix(char s[], int a, int b)
```

**【機能説明】**

int 型の値 a の -b 進表示した文字列を s に返す.

★ exercise-11-5 $\{b_i\}_{i=0}^{N}$ を 2 以上 36 以下の整数の列としたとき, unsigned int 型の値を $\{b_i\}_{i=0}^{N}$ が定める混合基数による表示の結果を表示するプログラムを以下のような関数を構成して書きなさい. ただし, $N$ は 32 以下とします.

**【形式】**

```
void mixed_radix(char s[], unsigned int a, int b[])
```

**【機能説明】**

unsigned int 型の値 a の b の定める混合基数表示した文字列を s に返す.

ただし, int 型は 32 ビットであることを, 文字列の長さを決めるために利用してもよい.

★ exercise-11-6 unsigned int 型の値をフィボナッチ表現した結果を求めるプログラムを以下のような関数を構成して書きなさい.

**【形式】**

```
int fibonacci_exp(int f[], unsigned int a)
```

**【機能説明】**

unsigned int 型の値 a のフィボナッチ表現の添字の列を f に返す. 戻り値は意味のある f の要素数です.

ただし, 以上の問題 (exercise-11-2 から exercise-11-6) において unsigned int, int 型は 32 ビットであることを, 文字列の長さを決めるために利用してもよい.

**【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.**





## ● 前回の講義のキーポイント及び補足

- 「配列」とは, **同じ型**の要素を**決まった個数**ならべたデータ構造である.
- 配列は異なる型の要素を並べることはできない. また, 配列の要素数は**定数式**で与える必要があり, コンパイル時に要素数が決定される必要がある.
- 関数に配列を渡すときには, 呼び出された関数側からは許される要素数を知ることはできない.
- 関数に配列を渡し, 関数内で配列要素を変更すると, 呼び出し側に戻ったときにもその変更が反映される. （単純型の場合には, 呼び出し側に戻ったとき, その変更は反映されない.）
- 文字列とは char 型の配列であり, 文字列の終端をあらわす \000 という文字（「ヌル文字」）が最後に付加されている.

★ ex10-2.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】**

```
int inner_product(int a[], int b[], int n)
```

**【機能説明】**

要素数 n を持つ２つの int 型の配列 a, b に対して, それをベクトルと思って, その内積の値 $\sum_{i=0}^{n-1} a_i b_i$ を返す.

```
/* $Id: ex10-2.c,v 1.1 2004-06-12 15:14:14+09 naito Exp $ */
#include <stdio.h>
int inner(int [], int [], int) ;
int main(int argc, char **argv)
{
  int   a[]={1,2,3}, b[3]={1,1} ;       /* ダメ！ b[3] = {1,1,0} と書くべき */
  printf("%d\n", inner(a,b,3)) ;
  printf("%d\n", inner(a,b,10)) ;       /* 一体何がおこるのか？ */
  printf("%d\n", inner(a,b,10000)) ;    /* 一体何がおこるのか？ */
  return 0 ;
}
/* 整数ベクトルの内積を求める.
 * 要素数は n */
int inner(int a[], int b[], int n)
{
  int   i, result=0 ;
  for(i=0;i<n;i++) result += a[i]*b[i] ;
  return result ;
}
```

ここで inner(a,b,10), inner(a,b,100000) を実行すると, 「意味不明な数値」が表示されたり, Segmentation Fault と表示されたりする. a, b ともに要素数 3 で定義されているため, a[4], b[4] を参照するとどのような値となるかは不定である. そのため, 「意味不明な数値」が表示される.

そのアプリケーションがアクセスすることを許可されていないメモリ領域にアクセスした場合に, OS から「例外割り込み」が発生し, プログラムの実行停止を要求される. そのような場合に発生する「例外割り込み」が「Segmentation Fault」である. つまり, 余りに大きな n に対して a[n] を参照するとこのような例外割り込みが発生する.

いずれにしても, 配列の要素数を越えたアクセスも可能であるため, 配列の要素数の範囲内でアクセスが行われるように注意すべきである.

★ ex10-5.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】**

```
void _strcpy(char t[], char s[])
```

**【機能説明】**

文字列 s を文字列 t にコピーする. t に s をコピーするだけの十分な要素数があるかどうかは配慮しない.

```
/* $Id: ex10-5.c,v 1.2 2004-06-12 17:37:23+09 naito Exp $ */
#include <stdio.h>
void _strcpy(char [], char[]) ;
int main(int argc, char **argv)
{
  char s0[]="This is a test string." ;
  char s1[]="これはテスト." ;
  char t0[30] ; /* s0, s1 をコピーするために十分な長さの文字列領域を確保 */
  char t1[5] ;  /* ためしに短い文字列領域を取ってみる */
  _strcpy(t0,s0) ; printf("%s\n", t0) ;
  _strcpy(t0,s1) ; printf("%s\n", t0) ;
  _strcpy(t1,s0) ; printf("%s\n", t1) ; printf("%s\n", t0) ;
  return 0 ;
}
/* 文字列 s を t にコピーする */
void _strcpy(char t[], char s[])
{
  int   i=0 ;
  while((t[i] = s[i])) i++ ;
  return ;
}
```

この結果は

```
This is a test string.
これはテスト.
This is a test string.
tring.
```

のようになったと考えられる. ここで, その理由を考えてみよう.





問題は _strcpy(t1,s0) の結果である. s0 には文字列長 22 の文字列が入っているが, t1 は要素数 5 の配列となっている. したがって, s0 を t1 にコピーする際に, t1 に許された要素数を越えてコピーが行われている.

実際には処理系・最適化・実行環境に依存するため, 必ずしもいつもこのようになるとは限らないが, t0, t1 は次のように並んでいると考えられ, t1 の先頭から２３バイトをコピーすることによって上のような結果が得られる.

| t1 | (空き) | t2 |
|---|---|---|
| 5バイト | １１バイト | ３０バイト |
| This␣ | is␣a␣test␣st | ring.\000s␣a... |

つまり, 配列の許された領域を越えてアクセスが発生することがある.

## ● 前回の課題の解説

★ exercise-10-1 ex10-1.c を閏年の場合にも正しい値を返し, 存在しない日付の場合には -1 を返すように改良しなさい. なお year は１９７０年以降であるとします. すなわち, year として 1969 以下の値が与えられたときは「存在しない日付」と考えます. （これは, 以下同じ主旨の問題に対して同様とします.）

```
/* $Id: exercise10-1-1.c,v 1.7 2004-06-17 12:53:42+09 naito Exp $ */
#include <stdio.h>
int     days_of_year(int, int, int) ;
int     is_leapyear(int year) ;
int main(int argc, char **argv)
{
  int   i ;
  for(i=0;i<12;i++)
    printf("%d\n", days_of_year(2000,i+1, 1)) ;
  printf("%d\n", days_of_year(2000,12,31)) ;
  return 0 ;
}
/* year 年 month 月 day 日が何日目かを返す. */
int days_of_year(int year, int month, int day)
{
  static int    n_days_of_month[]={31,28,31,30,31,30,31,31,30,31,30,31} ;
  static int    l_days_of_month[]={31,29,31,30,31,30,31,31,30,31,30,31} ;
  int           days = 0 ;
  int           i ;
  if ((month < 1)||(month > 12)) return -1 ;
  if (is_leapyear(year)) {
    if (day > l_days_of_month[month-1]) return -1 ;
    for(i=0;i<month-1;i++) days += l_days_of_month[i] ;
  }
  else {
    if (day > n_days_of_month[month-1]) return -1 ;
    for(i=0;i<month-1;i++) days += n_days_of_month[i] ;
  }
  days += day ;
  return days ;
}
int is_leapyear(int year)
{
  return ((year%4 == 0)&&(year%100 != 0))||(year%400 == 0) ;
}
```

- このプログラムは「本質的に同じ内容」を２回書いているという意味で, 冗長なプログラムである. （閏年の時の処理のために冗長になっている.）これは改良可能なのだが, これを改良するためには次回に解説する「ポインタ」を利用する必要がある.
- 閏年の処理のために,

  ```
  days_of_month[1] += 1
  ```

  とやってはいけない. なぜなら, days_of_month は static 宣言されているので, 次回以降に呼び出すときに配列要素の値が変更されたままになる.





★ exercise-10-2 西暦 year 年 month 月 day 日 hour 時 min 分 sec 秒が, その年の１月１日０時０分０秒から数えて何秒目になるかを返すプログラムを書きなさい. ただし, 存在しない日時の場合には, 何らかの形で呼び出し側からそれが検出できるようにしなさい.

```
/* $Id: exercise10-2-1.c,v 1.3 2004-06-17 13:45:20+09 naito Exp $ */
#include <stdio.h>

unsigned int    secs_of_year(int, int, int, int, int, int) ;
int             days_of_year(int, int, int) ;
int             is_leapyear(int year) ;

int main(int argc, char **argv)
{
  printf("%u\n", secs_of_year(2000,1,1,0,0,0)) ;
  printf("%u\n", secs_of_year(2000,12,31,23,59,59)) ;
  return 0 ;
}

/* 年内の通算秒数を返す */
unsigned int secs_of_year(int year, int month, int day,
                          int hour, int min, int sec)
{
  int           days, i ;
  unsigned int  secs ;

  days = days_of_year(year, month, day) ;
  secs = (days-1)*60*60*24 ;
  for(i=0;i<hour;i++) secs += 60*60 ;
  for(i=0;i<min;i++)  secs += 60 ;
  secs += sec ;
  return secs ;
}
```

ただし, is_leapyear, days_of_year は exercise-10-1 のものをそのまま流用する.

★ exercise-10-20 （**難かもしれない**）ex10-7.c で作った「文字列反転関数」を EUC-JP 日本語文字コードによる日本語文字列に対応するように拡張しなさい. なお EUC-JP 日本語は, ２バイトで日本語文字１文字をあらわし, その上位・下位バイトともに MSB （最上位ビット）は 1 となっている.

これによって文字列の要素が ASCII 文字なのか, EUC-JP 日本語文字コードの構成要素なのかを判定することができる.

このプログラムのめんどうなところは, EUC-JP 日本語文字コードが２バイトからなっていて, 文字列の中に「１バイト＝１文字」と［２バイト＝１文字」が混在していることである. そのため, 移動した文字が EUC-JP 日本語文字コードの第一バイトか第二バイトかを判断した後, 必要ならバイト列の入れ替えを行う必要がある. 文字列の入れ替えが終了した時点でも, EUC-JP 日本語文字コードのバイト列の入れ替えが残っている場合がある. しかも, 全体が奇数バイトか偶数バイトかによって状況が異なっていることに注意しなければならない.

具体的な例としては,

- “これは␣テ␣ス␣ト.” 文字列長が偶数で, 最後に入れ替えが必要.
- “これはテスト” 文字列長が偶数で, 最後に入れ替えが不要.
- “これはテスト.” 文字列長が奇数で, 最後に左の入れ替えが必要.
- “␣これはテスト” 文字列長が奇数で, 最後に右の入れ替えが必要.

等の例をテストしなければならない.





```
void _strrev(char s[])
{
  int           l=0, r ;
  int           is_l_euc = 0, is_r_euc = 0 ;
  char          c ;
  r = strlen(s) - 1 ;
  while(l < r) {
    /* 移動させるものが EUC-JP の構成要素か？ */
    /* EUC-JP の第一バイトなら is_?_euc = 1, 第二バイトなら is_?_euc = 2 */
    if (iseuc(s[r])) {
      if (is_l_euc == 0) is_l_euc = 1 ;
      else if (is_l_euc == 1) is_l_euc = 2 ;
    }
    else is_l_euc = 0 ;
    if (iseuc(s[l])) {
      if (is_r_euc == 0) is_r_euc = 2 ;
      else if (is_r_euc == 2) is_r_euc = 1 ;
    }
    else is_r_euc = 0 ;
    /* 実際の文字の移動
     * EUC-JP のバイト列入れ替えは第二バイトの移動後に行う */
    c = s[r] ; s[r] = s[l] ; s[l] = c ;
    if (is_l_euc == 2) {
      c = s[l] ; s[l] = s[l-1] ; s[l-1] = c ; is_l_euc = 0 ;
    }
    if (is_r_euc == 1) {
      c = s[r] ; s[r] = s[r+1] ; s[r+1] = c ; is_r_euc = 0 ;
    }
    l++; r-- ;
  }
  /* 最後に残っている可能性：
   * is_l_euc = 1 or is_r_euc = 2 の時, EUC-JP のバイト列入れ替えが終わっていない.
   * 1. 全体が奇数バイトの時, すなわち l=r の時：
   *    1-a. is_l_euc=1 の時：
   *         s[l-1] = 第二バイト, s[l] = 第一バイト： 入れ替え.
   *    1-b. is_r_euc=2 の時：
   *         s[r] = 第二バイト, s[r+2] = 第一バイト： 入れ替え.
   * 2. 全体が偶数バイトの時, すなわち l>r の時：
   *    入れ替えの必要があるのは, is_l_euc = 1 and is_r_euc = 2 の場合であり,
   *    s[l] と s[r] を入れ替える.                                          */
  if (l == r) {
    if (is_l_euc == 1) {
      c = s[l] ; s[l] = s[l-1] ; s[l-1] = c ;
    }
    if (is_r_euc == 2) {
      c = s[r] ; s[r] = s[r+1] ; s[r+1] = c ;
    }
  }
  else {
    if ((is_l_euc == 1)&&(is_r_euc == 2)) {
      c = s[l] ; s[l] = s[r] ; s[r] = c ;
    }
  }
  return ;
}
/* 日本語 EUC の構成バイトがどうか */
int iseuc(unsigned char s)
{
  return s&0x80 ;
}
```