● 今後の予定

【C言語（必ずやること）】

- ポインタ
- ポインタや配列を引数とした関数
- 文字列の基本操作
- 再帰的関数呼び出し
- ファイル入出力
- main の引数の意味
- 文字列から数値への型変換

【C言語（時間があったらやること）】

- 関数へのポインタ
- 多重配列とポインタ
- 構造体
- 動的なメモリ確保
- 自己参照構造体
- データ構造（スタック・リスト）

【アルゴリズム（必ずやること）】

- エラトステネスのふるい
- 基数変換

【アルゴリズム（時間があったらやること）】

- 公開鍵暗号
- 乱数の生成
- 数式の中置記法とポーランド記法

● 今日の実習

【サンプルプログラム】

★ ex10-1.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】

```
int days(int year, int month, int day)
```

【機能説明】

西暦 year 年が平年の場合, month 月 day 日がその年の１月１日から数えて何日目かを返す. ただし, year 年が閏年の場合には正しい値を返すとは限らない.

この中で, 次のことに注意してもらいたい.

- 関数内で days_of_month を static と宣言しているのは, １回目の関数呼び出し時にのみ「初期化」を行い, ２回目の呼び出し以降は「初期化」を省略するためである.

★ 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.

この中で, 次のことに注意してもらいたい.

- 関数仮引数は a[], b[] という形をしていて, そこには「配列の要素数」は明示されていない.
- 「配列の要素数」 n は, それを越えて配列要素へのアクセスが発生しないためだけに用いられている.

- 配列の初期化の方法で
  - “a[]={1,2,3}” とした場合には, 要素数は「初期化子」の個数となる.
  - “b[3]={1,1}” とした場合には, 要素数は配列宣言に規定された個数となり, 対応する「初期化子」が無い要素に対しては 0 で初期化される. しかし, このような中途半端な初期化は**やってはいけない**.
- このプログラム中には意図的に inner(a,b,10), inner(a,b,10000) という, 配列の要素数を越えたアクセスが行われている. この時どのようなことが起るのかをためし, それはなぜかを考えてみよう.

★ ex10-3.c このプログラムは配列要素をデータの列と考え, その平均値を求めている.

★ ex10-4.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 _strlen(char s[])
```

【機能説明】

文字列 s の長さを返す.

- 「文字列」の長さを返す関数は標準関数の中に strlen がある. その仕様は
  size_t strlen(const char s[])
  である. ここで型名 size_t に関しては下の「注意」を参照のこと.
  また const と宣言されたオブジェクトは「変更不可能」となる.

★ 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.

- 「文字列のコピー」を行う関数は標準関数の中に strcpy, strncpy がある.
- これまでは, 関数に渡された実引数は, **関数内部でそれを変更しても, 呼び出し側に戻ると変更は反映されない**と解説した. しかし, 実引数が「配列」の場合には, **関数内部でそれを変更すると, 呼び出し側でもその変更が反映される**. この理由は次回以降に解説するので, 今のところは「そんなもの」だと思っておこう.
- ここでは t1 として「わざと長さの足りない領域」を用意してコピーを行っている. この時, 他のオブジェクトはどうなってしまうのだろうか？

★ ex10-6.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 _strcapitalize(char s[])
```

【機能説明】

文字列 s に含まれる英文字の小文字を全て大文字にする.

- 各文字が「英文字の小文字」であるかどうかは, 標準関数 islower を用いて判定している. islower のような「文字種別判定関数」は引数の型として char ではなく int を取っていることに注意しよう.

- 大文字小文字変換は, 各文字に 'a' - 'A' を引くことで行っている. すなわち, 文字コードが ASCII であることを仮定している.

★ ex10-7.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 _strrev(char s[])
```

【機能説明】

文字列 s を逆転させる.

★ ex10-8.c このプログラムでは配列を使って（有限）集合の操作を行っている. 全体集合が N 個の要素からなるとき, それに $\{0,\ldots,N-1\}$ という番号をつけ, 配列でその部分集合を定義している. また, このプログラムの中では, 2つの部分集合 A, B に対して $A\cap B$ を求めている.

★ ex10-9.c ここでは以下のようなプログラムを書いている. int 型の要素数 N の配列 a を一変数多項式

$$a_0 + a_1x + a_2x^2 + \cdots + a_{N-1}x^{N-1}$$

と考え, その和を求める. ただし, 「桁あふれ」が生じると「イヤ」なので, $\mathbb{F}_2$ 係数であると考えている. （ここで, $\mathbb{F}_p$ とは位数 p の有限体をあらわす.）

- このプログラムが「イヤ」なところは, 各多項式の次数がわからないところである. そのため, 次数を含めたデータを用いる方が後々でうれしいことがある.
- 次数を含めたデータにするための方法は次の２つのやり方が考えられる.
  1. a[0] に「次数」を入れておき, 多項式の係数 a_i の値を a[i+1] に格納する方法. 簡単な方法だが, プログラムのループ制御が一見してわかりにくくなる欠点を持つ.
  2. 多項式を単純な「配列」としてあらわすことをあきらめ, 「係数」と「次数」からなる「構造を持つデータ」としてあらわす方法. このようなデータを（C言語では）「構造体」と呼び, 通常これを用いる.

★ ex10-10.c ここでは以下のようなプログラムを書いている. 利用している処理系の unsigned int または unsigned long であらわされる数値の範囲を越えて計算を行う. とりあえずここでは unsigned long を用いずに unsigned int の範囲を越えて計算を行うことを考えてみる. また, unsigned int は 32 ビットであることを仮定しておこう. そこで, 64 ビットの整数計算を行うために, 一つの数値を「unsigned int 型の要素数 2 の配列」に格納する.

このプログラムでは, そのように表現された 64 ビット整数の和を計算している. （ただし, 結果が 64 ビットを越えた場合の処理は行っていない.）

- 実際の数値 N の格納方法は以下の通り.
  - N の下位 32 ビットを a[0],
  - N の上位 32 ビットを a[1]
- 和は N の各ワード（各 32 ビット）を 16 ビットごとに和と繰上がりを計算することで得られる. また, 戻り値として最終的な繰上がりの値を返している.
- #define N 2 の 2 を 4 に取り替えれば, 自動的に $32N = 128$ ビットの計算が可能になる.

【注意】

- このサンプルプログラムの中では「配列の添字」をあらわす変数の型として int 型を用いている. しかし, より正確には「配列の添字」がどこまで使えるかは環境に依存し, そのため size_t という型名が定義されている. size_t は符号なし整数（より正確には unsigned int または unsigned long のいずれか）の「別名」として各環境で定義されている.

  したがって「配列の添字」をあらわす変数の型としては size_t を用いるのが正しい.
- 「配列の添字」をあらわす変数の型として「符号つき整数」を用いたプログラムを, その型として「符号なし整数」に変更すると, そのままではうまく動作しない場合が出てくる. ex10-10.c の print_uint32 関数がその例である. そこで用いられている変数 i を unsigned int にしたとき, 「どうしてうまく動作しなくなるのか？」, 「どこを変更すればよいのか？」を考えてみよう.

【課題】 以下の課題では, 必要に応じて関数を作ってプログラムを書くこと. また, 「標準関数」は自由に利用してかまわないが, 当然, その問題自身の目的になっている標準関数は利用してはならない. （必要なものは stdio.h, ctype.h, strings.h に含まれるものだけである.）

★ 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.）

★ 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.

★ exercise-10-3 西暦 year 年 month 月 day 日 hour 時 min 分 sec 秒が, 西暦１９７０年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.

★ exercise-10-4 N 個の int 型の配列要素であらわされたデータの分散を求めるプログラムを書きなさい.

★ exercise-10-5 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. 例えば, 8 の素因数分解の結果は $\{2,2,2\}$ であるとします. これは, 「全ての素因数を配列で戻してこい」と言っているのだが, int の最大値が 2^N-1 であることから, あらかじめ用意する配列の要素数が最大どれだけあるかはアプリオリに知ることができる.

★ exercise-10-6 配列に格納された int 型の数値の中から最大の数を選びだすプログラムを書きなさい.

★ exercise-10-7 （**ちょっとだけ難？**）与えられた文字列に含まれる単語の頭文字を大文字に変換するプログラムを書きなさい. ただし, 「単語」の区切り文字は「英数字」以外の文字とします.

★ exercise-10-8 次の仕様をみたす関数をつくりなさい.

【形式】

```
int arraycmp(int a1[], int a2[], int size)
```

【機能説明】

同じ要素数 size 個を持つ int 型の配列 a1, a2 に対して,

$$\sharp\{i \in [0, \mathtt{size}] : \mathtt{a1}[\mathtt{i}] \neq \mathtt{a2}[\mathtt{i}]\}$$

を返す.

★ exercise-10-9 与えられた文字列の中で指定の文字が最初にあらわれる場所が先頭から数えて何文字目かを求める関数, 与えられた文字列の中で指定の文字が最後にあらわれる場所が先頭から数えて何文字目かを求める関数を書きなさい. ただし, 先頭に指定の文字があらわれた場合には 0 を返すこととします. また, その文字が文字列中にあらわれない場合には -1 を返すこととします.

★ exercise-10-10 次の例のように, 与えられた文字列を右に N 文字回転させた文字列を出力するプログラムを書きなさい. ただし, N が負の場合, または文字列の長さを越えた場合には文字列を変換する必要はないものとします.

【例】 文字列が Nagoya␣University で $N = 5$ の場合には, rsityNagoya␣Unive とする.

★ exercise-10-11 ex10-8.c で使った方法を用いて, 集合の「共通部分」, 「対称差」, 「差」を求めるプログラムを書きなさい.

★ exercise-10-12 （**難**）$\mathbb{F}_2$ 係数の２つの一変数多項式 f, g に対して, その剰余を求めるプログラムを書きなさい.

★ exercise-10-13 （**難**）$\mathbb{F}_2$ 係数の２つの一変数多項式 f, g に対して, その最大公約多項式 $\gcd(f, g)$ を求めるプログラムを書きなさい.

★ exercise-10-14 （**難**）$\mathbb{F}_2$ 係数の２つの一変数多項式 f, g に対して, $fu + gv = \gcd(f, g)$, $fw + gz = 0$ をみたす多項式 u, v, w, z を求めるプログラムを書きなさい.

★ exercise-10-15 （**難**）exercise-10-12 から exercise-10-14 の内容を $\mathbb{F}_p$ 係数の一変数多項式に対して行うプログラムを書きなさい. ただし p は素数としますが, 処理系から量に依存するある上限の数 N よりも小さいものとします. この時, N としてはどのような値とするとプログラムが比較的簡単になるかを考えてプログラムしなさい.

★ exercise-10-16 （**難**）ex10-10.c で使った方法を用いて, 64 ビット整数の差を求めるプログラムを書きなさい.

★ exercise-10-17 （**もっと難**）ex10-10.c で使った方法を用いて, 64 ビット整数の積を求めるプログラムを書きなさい.

★ exercise-10-18 （**もっともっと難**）ex10-10.c で使った方法を用いて, 64 ビット整数の商・剰余を求めるプログラムを書きなさい.

★ exercise-10-19 （**もっともっともっと難**）ex10-10.c で使った方法を用いて, 64 ビット整数の平行根の整数部分を求めるプログラムを書きなさい.

★ exercise-10-20 （**難かもしれない**）ex10-7.cc 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.

【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.

★ ex10-1.c の内容

```
/* $Id: ex10-1.c,v 1.1 2004-06-12 15:14:16+09 naito Exp $ */
#include <stdio.h>
int     days_of_year(int, int, int) ;
int main(int argc, char **argv)
{
  int   i ;
  for(i=0;i<12;i++)
    printf("%d\n", days_of_year(2003,i+1, 1)) ;
  printf("%d\n", days_of_year(2003,12,31)) ;
  return 0 ;
}
/* year 年 month 月 day 日が何日目かを返す.
 * エラー処理なし. 平年のみ */
int days_of_year(int year, int month, int day)
{
  static int    days_of_month[]={31,28,31,30,31,30,31,31,30,31,30,31} ;
  int           days = 0, i ;
  for(i=0;i<month-1;i++) days += days_of_month[i] ;
  days += day ;
  return days ;
}
```

★ ex10-2.c の内容

```
/* $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 ;
}
```

★ ex10-3.c の内容

```
/* $Id: ex10-3.c,v 1.2 2004-06-12 18:02:32+09 naito Exp $ */
#include <stdio.h>
#define N       6
double mean(int [], int) ;
int main(int argc, char **argv)
{
  int   a[N] = {1,2,3,4,5,6} ;
  printf("mean = %f\n", mean(a,N)) ;
  return 0 ;
}
double mean(int a[], int n)
{
  int   total=0, i ;
  for(i=0;i<n;i++) total += a[i] ;
  return (double)total/n ;
}
```


★ ex10-4.c の内容

```
/* $Id: ex10-4.c,v 1.1 2004-06-12 15:14:11+09 naito Exp $ */
#include <stdio.h>
unsigned int _strlen(char []) ;
int main(int argc, char **argv)
{
  char s[]="This is a test string." ;
  printf("%u\n", _strlen("a")) ;
  printf("%u\n", _strlen(s)) ;
  printf("%u\n", _strlen("This is a test string.")) ;
  printf("%u\n", _strlen("これはテスト.")) ;
  return 0 ;
}
/* 文字列 s の長さを返す */
unsigned int _strlen(char s[])
{
  int   len = 0 ;
  while(s[len++]) ;
  return len-1U ;
}
```

★ ex10-5.c の内容

```
/* $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 ;
}
```


★ ex10-6.c の内容

```
/* $Id: ex10-6.c,v 1.2 2004-06-12 17:37:39+09 naito Exp $ */
#include <stdio.h>
#include <ctype.h>
void _strcapitalize(char []) ;
int main(int argc, char **argv)
{
  char s[]="This is a test string." ;
  _strcapitalize(s) ;
  printf("%s\n", s) ;
  return 0 ;
}

/* 文字列 s に含まれる英文字の小文字を全て大文字にする. */
void _strcapitalize(char s[])
{
  int   i=0 ;
  while(s[i]) {
    if (islower((int)s[i])) s[i] -= 'a'- 'A' ;
    i++ ;
  }
  return ;
}
```

★ ex10-7.c の内容

```
/* $Id: ex10-7.c,v 1.3 2004-06-12 18:03:49+09 naito Exp $ */
#include <stdio.h>
#include <strings.h>
void _strrev(char []) ;
int main(int argc, char **argv)
{
  char s[]="This is a test string." ;
  _strrev(s) ;
  printf("%s\n", s) ;
  return 0 ;
}
/* 文字列 s を逆転させる */
void _strrev(char s[])
{
  int   l=0, r ;
  char  c ;
  r = strlen(s) - 1 ;
  while(l < r) {
    c = s[r] ; s[r] = s[l] ; s[l] = c ;
    l++; r-- ;
  }
  return ;
}
```


★ ex10-8.c の内容

```
/* $Id: ex10-8.c,v 1.3 2004-06-12 15:32:51+09 naito Exp $ */
#include <stdio.h>
#define N       10

void    set_and(int [], int [], int []) ;
void    print_set(int []) ;

int main(int argc, char **argv)
{
  int   a[N] = {0,1,1,1,0,0,1,1,1,1} ;
  int   b[N] = {1,0,0,1,0,1,1,0,1,0} ;
  int   c[N] ;

  printf("a = {") ; print_set(a) ; printf("}\n") ;
  printf("b = {") ; print_set(b) ; printf("}\n") ;
  set_and(c,a,b) ;
  printf("c = {") ; print_set(c) ; printf("}\n") ;
  return 0 ;
}

/* C = A cap B */
void set_and(int c[], int a[], int b[])
{
  int   i ;

  for(i=0;i<N;i++) c[i] = a[i]&b[i] ;
  return ;
}

/* PRINT */
void print_set(int a[])
{
  int   i ;

  for(i=0;i<N;i++) {
    if (a[i]) printf("%d ", i) ;
  }
  return ;
}
```

★ ex10-9.c の内容

```
/* $Id: ex10-9.c,v 1.3 2004-06-12 17:37:56+09 naito Exp $ */
#include <stdio.h>
#define N       3
void polynomial_add(int [], int [], int []) ;
void polynomial_print(int []) ;
int main(int argc, char **argv)
{
  int   a[N]={1,1,0}, b[N]={1,0,1}, c[N] ;
  polynomial_add(c,a,b) ;
  polynomial_print(c) ; printf("\n") ;
  return 0 ;
}
/* F_2[x] の和 */
void polynomial_add(int c[], int a[], int b[])
{
  int   i ;
  for(i=0;i<N;i++) c[i] = a[i]^b[i] ;
  return ;
}
/* F_2[x] の表示 */
void polynomial_print(int a[])
{
  int   i, flag = 0 ;
  if (a[0]) {
    printf("1") ; flag = 1 ;
  }
  if (a[1]) {
    if (flag) printf(" + ") ;
    printf("x") ; flag = 1 ;
  }
  for(i=2;i<N;i++)
    if (a[i]) {
      if (flag) printf(" + ") ;
      printf("x^%d", i) ; flag = 1 ;
    }
  return ;
}
```


★ ex10-10.c の内容

```
/* $Id: ex10-10.c,v 1.1 2004-06-12 15:13:57+09 naito Exp $ */
#include <stdio.h>
#define N       2
#define L       (0x0000FFFF)
#define H       (0xFFFF0000)
#define HALFBITS        (16)
typedef unsigned int uint32 ;
void    print_uint32(uint32 []) ;
uint32  _sum(uint32 [], uint32 [], uint32[]) ;
int main(int argc, char **argv)
{
  uint32        a[N] = {0xFFFFFFFF, 0x00000000} ;
  uint32        b[N] = {0xFFFFFFFF, 0x00000001} ;
  uint32        c[N] ;
  _sum(c,a,b) ;
  print_uint32(a) ; print_uint32(b) ; print_uint32(c) ;
  return 0 ;
}
uint32 _sum(uint32 c[], uint32 a[], uint32 b[])
{
  uint32        al, ah, bl, bh, cl, ch, carry=0U ;
  int           i ;
  /* 各ワードの和を計算する */
  for(i=0;i<N;i++) {
    al=a[i]&L ; ah=(a[i]&H) >> HALFBITS ;
    bl=b[i]&L ; bh=(b[i]&H) >> HALFBITS ;
    cl = al + bl + carry ;
    carry = (cl&H) >> HALFBITS ; cl &= L ;
    ch = ah + bh + carry ;
    carry = (ch&H) >> HALFBITS ; ch &= L ;
    c[i] = (ch << HALFBITS) + cl ;
  }
  return carry ;
}
void print_uint32(uint32 a[])
{
  int   i ;
  for(i=N-1;i>=0;i--) printf("%08X", a[i]) ;
  printf("\n") ;
  return ;
}
```