担当：　富井尚志　(tommy@ynu.ac.jp)

第５回「配列を扱うアルゴリズム（２）」

前回　第４回「配列を扱うアルゴリズム（１）」の復習

第４回「配列を扱うアルゴリズム（１）」

☆ 配　列　（再出）

・配列って？，宣言の方法，計算機内部の記憶と配列の関係，多次元配列，配列の初期化，文字列

☆配列を用いたプログラム例

◎ 線形探索

・探索，線形探索，通常のプログラム（lsearch1.c），番人（sentinel）ありのプログラム（lsearch2.c）

◎ 2 分探索

・ある区間のちょうど真中に位置するデータを調べる．（bsearch.c）

◎ 最大値(最小値)

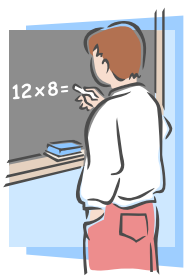
・1 番め ～（i-1）番め までの最大値を(max)とし，i = 2 から n まで変化させて調べる．（max.c）

◎ 前回の出席票の小テストの解答

練習問題：　２次元配列を用いて九九の表を画面に表示する次のプログラム中の［　　］内に適切な記述を補足せよ．

```
include<stdio.h>

int ku_ku[ 9 ][ 9 ]; /* 九九の値を代入する２次元配列 */

int main(void)
{
    int i, j;

    /* 値の代入 */
    for ( i=0; i< 9 ; i++ )
        for ( j=0; j< 9 ; j++ )
            ku_ku[i][j] = ( i + 1 ) * ( j + 1 ) ;
    /* 九九の表の表示 */
    printf("x | 1  2  3  4  5  6  7  8  9¥n");
    printf("---------------------------¥n");
    for ( i=0; i< 9 ; i++ ){
        printf("%d",i+1 );
        for ( j=0; j< 9 ; j++ )
            printf("%2d ", ku_ku[i][j] );
        printf("¥n");
    }
    return 0;
}
```

ku_ku[0,0], ku_ku[0,1], …, ku_ku[0,8],
ku_ku[1,0], ku_ku[1,1], …, ku_ku[1,8],
・・・・・・・・・・・・・・・・・・・・
ku_ku[8,0], ku_ku[8,1], …, ku_ku[8,8]
が定義される．[ ]の中は要素の数.



第 5 回の始まり

配列を扱うアルゴリズム（2）とポインタ

☆ 集合を扱う

◎ 集合の表現

この集合ではない…数学の集合です

a. 整数（ビット列）の各ビットに各要素を割り当てる.
   - ビットの長さだけの要素を表せる.
   - 高速演算可能
   - 小規模なもの

b. 配列に各要素を代入する.

◎ 配列を使った表現

- 一つの配列を一つの集合としてみなす.
- 集合は要素の数が変化するので,
  1. 十分な大きさの配列
  2. 配列のどの部分までが要素で埋められているかを表す方法

  が必要

◎「要素の列」(bag)を「集合」(set)にする.

- 重複要素の削除　{ 1, 3, 1, 2, 3, 5 }　→　{ 1, 3, 2, 5 }
- テストプログラム setop.c 内の 関数 rmdup( )

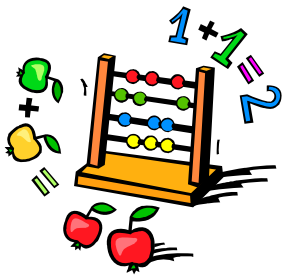
◎ 要素（member)

例:　2 ∈ { 1, 2, 4, 5} → 真　　3 ∈ { 1, 2, 4, 5 } → 偽

- 線形探索
- テストプログラム setop.c 内の 関数 set_member( )

◎積（intersection)

例: { 1, 2, 4, 5 } ∩ { 1, 3, 5, 6 } → { 1, 5 }

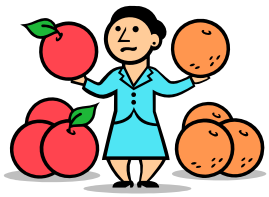


- 新しい集合 C(空集合)を用意. A の要素を一つずつ調べ,
  集合 B の要素なら，集合 C に加える.
- テストプログラム setop.c 内の 関数 set_intersec( )

◎和（union)

例: { 1, 2, 4, 5 } ∪ { 1, 3, 5, 6 } → { 1, 2, 3, 4, 5, 6 }

- 新しい集合 C(空集合)を用意. A の要素を一つずつ調べ, 集合 B の要素でないなら,
  集合 C に加える．最後に C に B の要素すべてを加える.
- テストプログラム setop.c 内の 関数 set_union( )



◎差（difference)

例: { 1, 2, 4, 5} ＼ { 1, 3, 5, 6} → { 2, 4 }

- 新しい集合 C(空集合)を用意. A の要素を一つずつ調べ, 集合 B の要素でないなら, 集合 C に加える.
- テストプログラム setop.c 内の 関数 set_difference( )

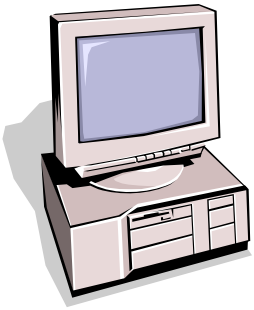
☆ 文字列(string)

◎ C 言語における文字列

- 特別な型はない.
- 文字(char)型の一次元配列であり，文字 '¥0' が文字列の最後を表す.
- 例: char str[ ] = "Taro";
  → str[0]=='T', str[1]=='a', str[2]=='r', str[3]=='o', str[4]=='¥0'
  つまり，次の初期化と同じ.
  char str[ ] = { 'T', 'a', 'r', 'o', '¥0' };
- 特殊文字
  '¥0' → ヌル文字(値 0)，'¥n' → 改行，'¥t' → 水平タブ など

◎ 文字列を扱うプログラム

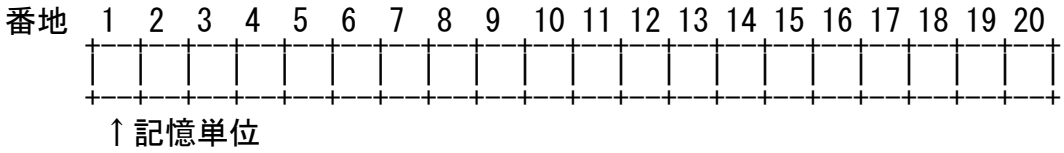
例: 文字列の連結



- テストプログラム strcat.c の関数 strcat(s,t)
- 文字列 s の末尾に文字列 t を連結する.

☆ 計算機の記憶とポインタ

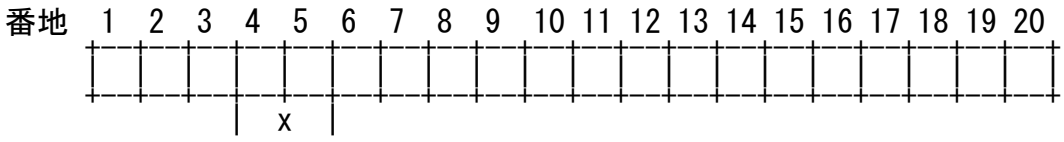
◎ 計算機の記憶



- ある記憶単位が 1 列に並んでいる.
- 記憶単位には一意に識別できる番地(アドレス)がふられている.
  アドレス = 記憶場所

◎（プログラム中の）データを計算機の記憶上に結びつけるには...

- 記憶場所の最初のアドレス
- （記憶単位で計った）データの大きさが分かればよい．例えば， double x;
  と宣言されているとして，以下のように配置されているとすると...



4 番地から double の大きさの記憶領域を確保したデータ = 変数 x

◎ ポインタ

- ポインタ ＝（最初のアドレス ＋ 記憶領域の大きさ）からなるデータ.
- 使われる場所
  - a. call by reference の代わり
    - ... 呼出し側の変数の値を（呼び出された）関数側で変更
      - → 引数を使って**複数の出力**を得る関数も作れる（練習問題参照）
  - b. 記憶領域を直接扱う動的なデータ構造（後に扱う）

- 例 1　int *xp;

xp は整数型のデータへのポインタを格納する変数（中身はアドレス）
（整数型データを格納する（している）記憶場所の最初のアドレス）

単項演算子 ＊ を アドレス に適用するとその 変数 になる
単項演算子 ＆ を 変数 に適用するとその アドレス になる

非常に重要

- 例 2　int *xp, x, y;

整数型データを指すポインタ xp の宣言
int *xp は，本来 (int *)xp の意味で，単項演算子 * を施すと int 型をさすポインタの意です.

```
x  =  3;
xp  =  &x;     x の先頭アドレスを整数型へのポインタ変数 xp に代入.
y  =  *xp;     xp（＝x へのポインタ）の指す値（=x の値（=3））を y に
               代入．結局 y=3 となる
```

変数 x の先頭アドレスを意味する

この数行はポインタの理解に非常に重要です．各行で行われている内容についてよく考え，しっかり理解して下さい．さもないと，後で混乱することになります！

- 例 3 「call by reference の代わり」
  テストプログラム pointer.c

```
/*************************************************************
    アルゴリズムとデータ構造
    サンプルプログラム　setop.c
    <<集合演算>>
    copyright (c) 1995,96,97   T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/
#include <stdio.h>

#define MAX          100    /* MAX は 配列の最大値要素数 */
#define EOSET         -1    /* 集合の終りを表すマーク */
#define NOTMEMBER     -2    /* 集合の要素がないことを表すマーク */

#define TRUE           1     /* 真 */
#define FALSE          0     /* 偽 */

void rmdup(int a[ ]);
void printset(int a[ ]);
void set_intersec(int a[ ],int b[ ], int c[ ]);
void set_union(int a[ ],int b[ ], int c[ ]);
void set_difference(int a[ ], int b[ ], int c[ ]);

int main(void)
{
  int s[MAX] = {3,4,2,6,2,9,9,9,1,5,2,9,6,8,6,EOSET};
  int t[MAX] = {1,2,3,4,7,8,9,10,EOSET};
  int u[MAX] = {1,  3,4,5,6,7,9,EOSET};
  int v[MAX];

  printf("s[]: ");
  printset(s);
      /* 配列の名前だけを指定すると配列自身を指定したことになる.
       * 正確にいうと，配列が配置されている記憶領域の先頭の番地
       * を表す.
       */

  /* 重複除去 */
  printf("rmdup(s)¥n");
  rmdup(s);
  printf("s[]: ");
  printset(s);

  printf("t[]: ");
  printset(t);
  printf("u[]: ");
  printset(u);

  /* 集合の積 (交わり) */
  printf("set_intersec(t,u,v)¥n");
```

これらの関数の引数として int 型の配列の先頭番地（＝&a[0], &b[0], &c[0]＝a, b, c）を渡しています（配列名は配列の先頭アドレスを指します）.

関数 printset に，整数型配列 s[0]～s[99]の先頭番地 s（＝&s[0]＝s）を渡している（以下同様）.

```
49    set_intersec(t,u,v);
50    printf("v[]: ");
51    printset(v);
52
53    /* 集合の和 (結び) */
54    printf("set_union(t,u,v)¥n");
55    set_union(t,u,v);
56    printf("v[]: ");
57    printset(v);
58
59    /* 集合の差 */
60    printf("set_difference(t,u,v)¥n");
61    set_difference(t,u,v);
62    printf("v[]: ");
63    printset(v);
64    return 0;
65  }
66
67
68  /* 重複を除去する関数 */
69  void rmdup(int a[])
70  {
71    int i,j,d;
72
73    /* 重複する要素にマークをつける */
74    for(i=0; a[i] != EOSET; i++)
75      for(j=i+1; a[j] != EOSET; j++)
76        if (a[j] == a[i])
77            a[j] = NOTMEMBER;
78
79    /* マークのついた要素をつめるように配列を作り直す */
80    d = 0;
81    for(i=0; a[i] != EOSET; i++)
82      if (a[i] == NOTMEMBER)
83        d++; /* 要素の移動距離を 1 だけ増やす */
84      else {
85        if (d > 0)
86            a[i-d] = a[i]; /* 距離 d だけ要素を前方に移動 */
87      }
88    a[i-d] = a[i]; /* 配列の最後のマークも移動 */
89  }
90
91
92  /* 集合の印刷 */
93  void printset(int a[])
94  {
95    int i;
96    for (i = 0; a[i] != EOSET; i++) {
```

この直線は関数の境界を分かり易くするためのものです.

i	0	1	2	3	4	5	...
a[i]	3	4	2	6	2	9	...
j	...	...	...	3	4	5	...
a[j]	...	...	...	6	2	9	

a[ ]	3	4	2	6	NOT MEMBER	9	...

a[ ]	3	4	2	6	NOT MEMBER	9	NOT MEMBER	NOT MEMBER	1	...

d=1　　d=3

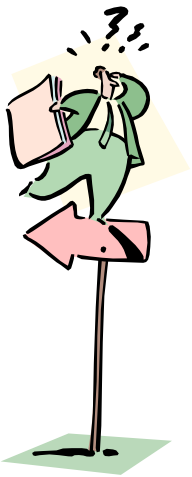
a[ ]	3	4	2	6	9	1				...

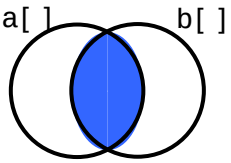
ここでの配列 a[ ]は, この関数を呼ぶ側でどの配列を引数に入れるかで異なる. 例えば行番号 30 で呼ばれたときは s[ ]のことなので次のようになる.

a[0]	a[1]	a[2]	a[3]	...	a[15]	...
3	4	2	6	...	EOSET	...

EOSET を見つけたらこの for ループが終了する.

```
97        printf("%d ",a[i]);
98      }
99      printf("¥n");
100   }
101
102
103   /* 集合の要素判定 */
104   /* 第一引数の整数が，第二引数の配列の中にあれば 真 */
105   /* そうでなければ 偽 を返す */
106   int set_member(int x, int a[])
107   {
108     int i=0;
109
110     /* 線形探索 */
111     while(a[i] != x && a[i] != EOSET)
112       i++;
113
114     if(a[i] == x)
115       return TRUE;
116     else
117       return FALSE;
118   }
119
120
121   /* 集合の積 */
122   /* 第一引数の配列と第二引数の配列に共通して含まれる要素を */
123   /* 第三引数の配列に入れる */
124   void set_intersec(int a[], int b[], int c[])
125   {
126     int i,j;
127
128     for(i=0,j=0; a[i] != EOSET; i++)
129       if (set_member(a[i],b)) {
130         c[j]=a[i];
131         j++;
132       }
133     c[j] = EOSET;
134   }
135
136
137   /* 集合の和 */
138   /* 第一引数の配列と第二引数の配列のうち */
139   /* すくなくともどちらか一方に含まれる要素を */
140   /* 第三引数の配列に入れる */
141   void set_union(int a[], int b[], int c[])
142   {
143     int i,j;
144
```

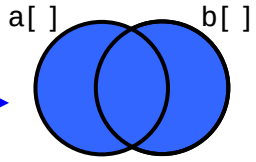




この例のように for ループ中の式は復文でも良いが，慣れないうちは使わない方が無難である．

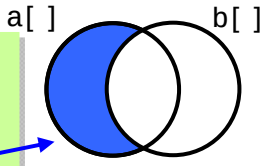
“もしも b[ ]の中に a[i]と同じ要素があれば，c[j]にその要素を代入しなさい”

c[ ]の最後に EOSET(=-1)を代入している．

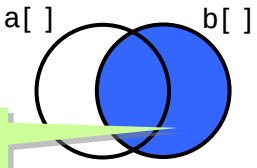


```
145     for(i=0,j=0; a[i] != EOSET; i++)
146       if (! set_member(a[i],b)) {
147         c[j]=a[i];
148         j++;
149       }
150
151     for(i=0; b[i] != EOSET; i++, j++)
152       c[j]=b[i];
153
154     c[j]=EOSET;
155   }
```

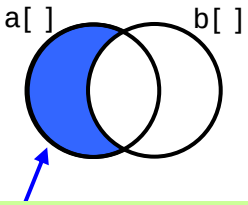
“もしも b[ ]の中に a[i]と同じ要素がなければ，c[j]にその要素を代入しなさい” つまりこの部分を求めている.



この部分(=b[ ])を c[ ]に追加し，最後に EOSET を代入している.




```
157
158   /* 集合の差 */
159   /* 第一引数の配列と第二引数の配列のうち */
160   /* 第一引数の配列だけに含まれる要素を */
161   /* 第三引数の配列に入れる */
162   void set_difference(int a[], int b[], int c[])
163   {
164     int i,j;
165
166     for(i=0,j=0; a[i] != EOSET; i++)
167       if (! set_member(a[i],b)) {
168         c[j]=a[i];
169         j++;
170       }
171
172     c[j]=EOSET;
173   }
174
```



行番号 149～153 と同じで，この部分を求めて c[ ]に代入している.

175 【実行結果】

176 s[]: 3 4 2 6 2 9 9 9 1 5 2 9 6 8 6

177 rmdup(s)

178 s[]:

179 t[]:

180 u[]:

181 set_intersec(t,u,v)

182 v[]:

183 set_union(t,u,v)

184 v[]:

185 set_difference(t,u,v)

186 v[]:

```
/*************************************************************
     アルゴリズムとデータ構造
     サンプルプログラム　strcat.c
     <<文字列を扱う例---連結>>
     copyright (c) 1995,96,97　T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/

#include <stdio.h>

#define MAX 20
void strcat(char s[], char t[]);

int main(void)
{
  char x[MAX] = "Hello, ";
  char y[MAX] = "world.";

  printf("x[] == %s¥n", x);
  printf("y[] == %s¥n", y);
  strcat(x,y);
  printf("strcat(x,y)...done.¥n");
  printf("x[] == %s¥n", x);
  return 0;
}

/* 文字列 s の末尾に 文字列 t を連結する */
void strcat(char s[], char t[])
{
  int i=0,j=0;

  /* 文字列 s の末尾のインデクスを i の値とする */
  while(s[i] != '¥0')
    i++;
  /* s の末尾に t の内容をコピー */
  /* '¥0' もコピーされなければいけないことに注意 */
  while((s[i++] = t[j++]) != '¥0')
    ;
}
```

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]
H	e	l	l	o	,	¥0

y[0]	y[1]	y[2]	y[3]	y[4]	y[5]	y[6]
w	o	r	l	d	.	¥0

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]
H	e	l	l	o	,	¥0

t[0]	t[1]	t[2]	t[3]	t[5]	t[6]	t[7]
w	o	r	l	d	.	¥0

s[0]	s[1]	s[2]	s[3]	s[4]	s[5]	s[6]	s[7]	s[8]	s[9]	s[10]	s[11]	s[12]
H	e	l	l	o	,	w	o	r	l	d	.	¥0

s[ ]は（先頭番地がいっしょなので）x[ ]と同じものであり，結局，x[ ]を変えたことになる．

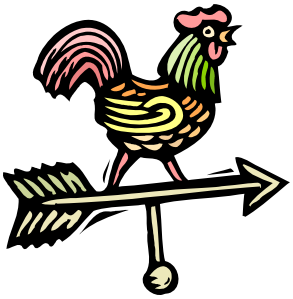
分かりにくいかも知れないけれど重要です！

【実行結果】

```
x[] == Hello,
y[] == world.
strcat(x,y)...done.
x[] == Hello, world.
```

```
/*************************************************************
    アルゴリズムとデータ構造
    サンプルプログラム　pointer.c
    <<アドレスとポインタ>>
    copyright (c) 1995,96,97　T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/

#include <stdio.h>

void f1(int x);
void f2(int *xp);

int main(void)
{
  int a;
  a = 1;
  printf("a == %d¥n", a);
  printf("start: f1(a)¥n");
  f1(a);
  printf("done:  f1(a)¥n");
  printf("a == %d¥n", a);
  /* &a は変数 a の値が保持されている番地 */
  /* printf の引数の中の%u は%d とほぼ同じだが符合なしの整数として印刷 */
  printf("&a == %u¥n", &a);
  printf("start: f2(&a)¥n");
  f2(&a);
  printf("done:  f2(&a)¥n");
  printf("a == %d¥n", a);
  return 0;
}

/* 変数の値を仮引数(自動変数)に代入し，そのデータを改変する */
/* 呼び出し側の変数の値はもちろん変わらない */
void f1(int x)
{
  printf("    x == %d¥n", x);

  x = x + 1;

  printf("    x == %d¥n", x);
}

/* 変数へのポインタ(番地)を仮引数に渡し，その番地にあるデータを改変する */
/* 呼び出し側の変数の値も変わる */
void f2(int *xp)
{
  printf("   xp == %u¥n", xp);
  printf("  *xp == %d¥n", *xp);
```

関数 f1 に渡している引数は　整数型の変数 x
関数 f2 に渡している引数は　整数型の変数へのポインタ
よく覚えておいて下さい.



自動変数 x はこの関数の内部だけで有効.
このため，ここで x の値を変化させても，この関数を呼んだ側の引数の値を変化させることはできない.

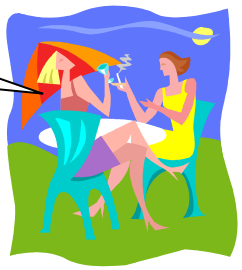
```
49
50      *xp = *xp + 1;
51
52      printf("    xp == %u¥n", xp);
53      printf("   *xp == %d¥n", *xp);
54  }
55
56
57  【実行結果】
58  a == 1
59  start: f1(a)
60          x ==
61          x ==
62  done:   f1(a)
63  a ==
64  &a == 4026529420d
65  start: f2(&a)
66          xp == 4026529420d
67         *xp ==
68          xp == 4026529420d
69         *xp ==
70  done:   f2(&a)
71  a ==
72
73
```

アドレス xp に保存されている整数型変数の値に 1 を加えて変化させることにより，**呼んだ側の変数の値も変わる**（そのアドレスの中身が変化するから．）

故意に消しています．どのような実行結果になるか考えましょう．

アドレスの値は計算機や試行により異なります．これは一例です．

焦らず，着実に理解しましょう．



番地	…	&a (=xp)	…	?	…
内容	…	1	…	1	…

実際は同じもの

main 中の a

f2 中の*xp

f1 中の x

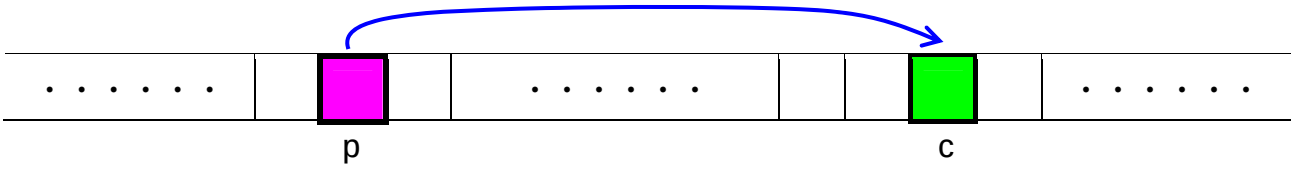
main 中で f2(&a); とすることにより，変数 a のアドレス&a（＝xp）を指すポインタを f2 に渡す．F2 で*xp とは，main の a と実は同じものなので，*xp を変化させると，main 中の a も変化する．

main 中で f1(a); とすることにより，どこかに確保された自動変数 x のアドレスに xp の内容がコピーされる．**関数 f1 中でいくら x を変化させても，元のアドレス xp の中身は変わらない**．

ポインタとアドレスに関する補足

ポインタがＣで頻繁に利用される理由：
1. それが時として計算を表現する唯一の方法であるため.
2. これで他の方法よりコンパクトで効率的なプログラムが書けるから.

例えば, c が char で, p がそれを指すポインタであればこの状況は次のように表される.



- **単項演算子 &** ：**変数のアドレスを与える.**
  例）p = &c;　　変数 p に c の**アドレス**が代入される
- **単項演算子 ＊** ：**間接演算子（逆参照演算子）. アドレスに適用すると変数を表す.**
  例）int x = 1, y = 2, z[10];
  int *ip;　　→ ip は int へのポインタである

重要！

①　ip = &x;　　→ ip は今 x を指す
②　y = *ip;　　→ y はこれで 1 となる
③　*ip = 0;　　→ x はこれで 0 となる
④　ip = &z[0];　　→ ip はいまは z[0] を指す