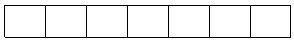
第４章 データ型

4.1 配列（array）

同じ型のデータを複数格納できる箱



配列の大きさ…格納できる要素の数

配列の宣言

```
int week[7];
```

week[0] [1] [2] [3] [4] [5] [6]

e_0	e_1	e_2	e_3	e_4	e_5	e_6

配列要素（element）の参照と代入
→ インデックスを使う．

```
week[1] = week[4];
```

week[0] [1] [2] [3] [4] [5] [6]

e_0	e_4	e_2	e_3	e_4	e_5	e_6

n 個（$0 < n \leq 100$）の正の整数を読み込み，入力と逆の順序で表示する

```
main()
{
    int a[100];
    int x, i = 0;

    for (;;) {
        scanf("%d", &x);
        if (x == 0) break;
        a[i++] = x;
    }
    while (i > 0)
        printf("%d ", a[--i]);

    printf("\n");
}
```

a[0][1][2] [99]

x_1				

a[0][1][2] [99]

x_1	x_2			

a[0][1] [$k-1$] [99]

x_1	x_2		x_k		

a[0][1] [$n-1$] [99]

x_1	x_2		x_n		

ソーティング（sorting）

小さい順に並べかえる

```
main()
{
    int a[100];
    int x, i, j, n, m, temp;

    n = 0;
    for (;;) {
        scanf("%d", &x);
        if (x == 0) break;
        a[n++] = x;
    }
    for (i = 0; i < n-1; i++) {
        m = i;
        for (j = i + 1; j < n; j++) {
            if (a[m] > a[j])
                m = j;
        }
        temp = a[m];
        a[m] = a[i];
        a[i] = temp;
    }
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);

    printf("\n");
}
```


以下を $i = 0$ から $i = n - 2$ まで繰り返す．
a[i] から a[$n-1$] までの最小値を探す．
最小値が a[m] に入っていたとする．
a[i] と a[m] の値を交換する．

$n = 4$ の例

a[0][1][2][3]

5	2	7	4		

$i = 0$, $m = 1$

a[0][1][2][3]

2	5	7	4		

$i = 1$, $m = 3$

a[0][1][2][3]

2	4	7	5		

$i = 2$, $m = 3$

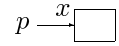
a[0][1][2][3]

2	4	5	7		

4.2 ポインタ（pointer）

変数の場所を表すデータ

「p は変数 x を指す」



このとき

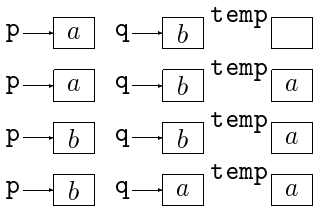
*p … x の値の参照

*p = e … x の値の変更

任意の 2 つの整数変数の内容を交換する

```
void swap(int *p, int *q)
{
    int temp;

    temp = *p;
    *p = *q;
    *q = temp;
}
```



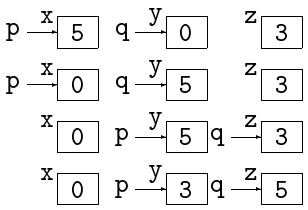
swap を呼び出す

```
main()
{
    int x = 5, y = 0, z = 3;

    printf("x = %d, y = %d, z = %d\n",
           x, y, z);
    swap(&x, &y);
    printf("x = %d, y = %d, z = %d\n",
           x, y, z);
    swap(&y, &z);
    printf("x = %d, y = %d, z = %d\n",
           x, y, z);
}
```

出力結果

```
x = 5, y = 0, z = 3
x = 0, y = 5, z = 3
x = 0, y = 3, z = 5
```



配列へのポインタ

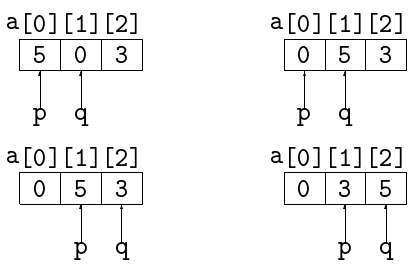
```
main()
{
    int a[3];

    a[0] = 5; a[1] = 0; a[2] = 3;

    printf("a = {%d, %d, %d}\n",
           a[0], a[1], a[2]);
    swap(&a[0], &a[1]);
    printf("a = {%d, %d, %d}\n",
           a[0], a[1], a[2]);
    swap(&a[1], &a[2]);
    printf("a = {%d, %d, %d}\n",
           a[0], a[1], a[2]);
}
```

出力結果

```
a = {5, 0, 3}
a = {0, 5, 3}
a = {0, 3, 5}
```



ポインタ演算

等号と不等号

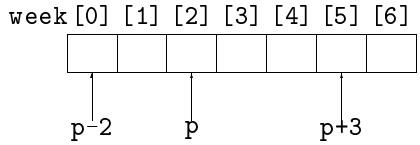
p_1==p_2　同じ変数を指しているか

p_1!=p_2　異なる変数を指しているか

加減算

p+n　後方 n 番目の変数へのポインタ

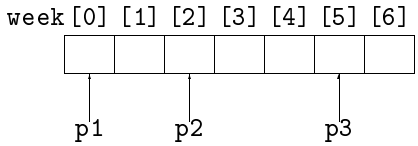
p−n　前方 n 番目の変数へのポインタ




```
*(p+3) = *(p-2);
```

引き算

p_1−p_2　ポインタの差



比較

p_1<p_2　$p_1 < p_2$ かどうか

p_1>p_2　$p_1 > p_2$ かどうか

p_1<=p_2　$p_1 \leq p_2$ かどうか

p_1>=p_2　$p_1 \geq p_2$ かどうか

代入とインクリメント

p_1=p_2	p_2 を p_1 に代入
p+=n	p+n を p に代入
p-=n	p-n を p に代入
p++	p を 1 増やす
++p	p を 1 増やす
p--	p を 1 減らす
--p	p を 1 減らす

ソーティング

```
void sort(int *from, int *to)
{
    int *p, *q, *m, temp;

    for (p = from; p < to; p++) {
        m = p;
        for (q = p + 1; q < to; q++) {
            if (*m > *q)
                m = q;
        }
        temp = *m;
        *m = *p;
        *p = temp;
    }
}
```


4.3 文字（character）と文字列（string）

文字データは整数（文字コード）で表現されている．
ほとんどの WS では，ASCII コードを使っている．

画面にまともに表示できる文字を表示する．

```
main()
{
    int c, n = 0;

    for (c = ' '; c <= '~'; c++) {
        printf("%c ", c);
        if (++n == 20) {
            printf("\n");
            n = 0;
        }
    }
    printf("\n");
}
```

出力結果

```
  ! " # $ % & ' ( ) * + , - . / 0 1 2 3
4 5 6 7 8 9 : ; < = > ? @ A B C D E F G
H I J K L M N O P Q R S T U V W X Y Z [
\ ] ^ _ ` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
```

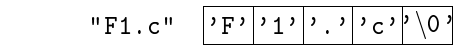
文字操作に関するライブラリ

```
isdigit(c)    islower(c)    isupper(c)
isalpha(c)    isalnum(c)
```


文字の種類を判定する．

```
#include <ctype.h>
void ctype(int c)
{
    if (isdigit(c))
        printf("numeric '%c'\n", c);
    else if (islower(c))
        printf("lower '%c'\n", c);
    else if (isupper(c))
        printf("upper '%c'\n", c);
    else
        printf("special '%c'\n", c);
}
```

文字列は要素が char 型の配列




```
main()
{
    char *s = "F1.c";
    do
        ctype(*s);
    while (*(++s) != '\0');
}
```

出力結果

```
upper 'F'
numeric '1'
special '.'
lower 'c'
```


4.4 構造体（structure）

複数のデータ（メンバ）から構成されるデータ

位置を表すデータ

x 座標
y 座標

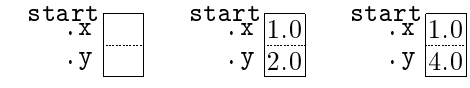
構造体の定義

```
struct point {
    double x;
    double y;
};
struct point start;
```

メンバの参照と代入

```
start.x = 1.0;
start.y = 2.0;

start.y = start.y * 2;
```



構造体全体の代入

```
struct point end;
...
end = start;
```

start: .x 1.0, .y 4.0　end: .x 1.0, .y 4.0

2点 $P(x_1, y_1)$ と $Q(x_2, y_2)$ の距離
$r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ を求める

```
double
distance(struct point P, struct point Q)
{
    double dx = P.x - Q.x;
    double dy = P.y - Q.y;

    return sqrt(dx * dx + dy * dy);
}
```

構造体へのポインタ

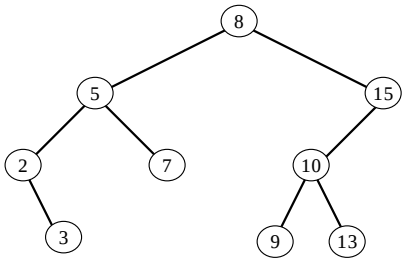
```
struct point *p;
p = &start;
```

p → 1.0, 4.0

```
*p
(*p).x
p->x
```


4.5 再帰的データ構造

二進検索木（binary search tree）



n を検索するには：

1. 節の数値 v が n と等しければ検索終了.
2. $n < v$ であれば左側の枝へ進む.
   もし左側の枝がなければ検索終了.
3. $n > v$ であれば右側の枝へ進む.
   もし右側の枝がなければ検索終了.

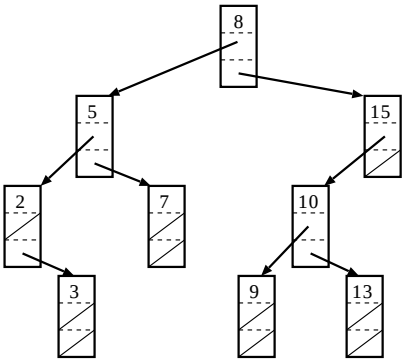
9 を探す：8 → 15 → 10 → 9 （成功）
1 を探す：8 → 5 → 2 （失敗）

二進検索木の節を定義する

```
struct node {
    int value;            /* 節の値 */
    struct node *left;    /* 左の枝 */
    struct node *right;   /* 右の枝 */
};
```

再帰的データ型（recursive data type）

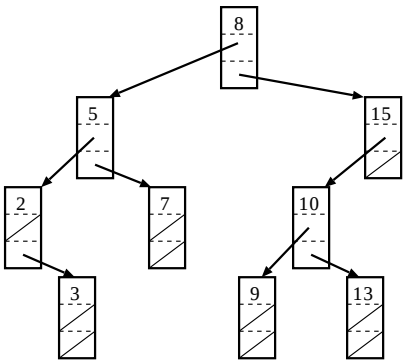
再帰的データ構造（recursive data structure）




```
int search(struct node *p, int n)
{
    if (p->value == n)
        return 1;
    else if (n < p->value)
        if (p->left != NULL)
            return search(p->left, n);
        else
            return 0;
    else
        if (p->right != NULL)
            return search(p->right, n);
        else
            return 0;
}
```

NULL は空ポインタ（null pointer）

ヒープ（heap）




```
struct node heap[100];
```