c言語

数値計算のためのc言語

c言語の特徴

- 低水準な処理も可能な高水準言語
- 汎用構造化言語
- オブジェクト指向言語(c++, java)の基礎
- ソフトウェア技術者の教養的知識
- 最新のライブラリ(過去の資産は使えない)
- 数値計算には不向き(FORTRANに比べて)

プログラムの基本構成

```
#include <stdio.h>
#include <math.h>
int main(void) {
  double x, y;
  printf("Hello, world.\n");
  x = 0.1;
  y = sin(x);
  printf("SIN(%lf) = %lf\n", x, y);
  return 0;
}
```

Linux等でコンパイル・実行する場合

```
$ gcc this_file.c -lm
```


プログラムの基本構成

```
#include <stdio.h>
#include <math.h
int main(void) {
  double x, y;
  printf("Hello, world.\n");
  x = 0.1;
  y = sin(x);
  printf("SIN(%lf) = %lf\n", x, y);
  return 0;
}
```

- 用意されているキーワード(予約語)

プログラムの基本構成

```
#include <stdio.h>
#include <math.h
int main(void) {
  double x, y;
  printf("Hello, world.\n");
  x = 0.1;
  y = sin(x);
  printf("SIN(%lf) = %lf\n", x, y);
  return 0;
}
```

- ユーザ定義名前付きオブジェクト
- ユーザ定義リテラル

プログラムの基本構成

```
#include <stdio.h>
#include <math.h>
int main(void) {
  double x, y;
  printf("Hello, world.\n");
  x = 0.1;
  y = sin(x);
  printf("SIN(%lf) = %lf\n", x, y);
  return 0;
}
```

- システムに依存するリテラル
  - 標準入出力ライブラリ
  - 標準数学ライブラリ

プログラムの基本構成

```
#include <stdio.h>
#include <math.h>
int main(void) {
  double x, y;
  printf("Hello, world.\n");
  x = 0.1;
  y = sin(x);
  printf("SIN(%lf) = %lf\n", x, y);
  return 0;
}
```

- プリプロセッサにより1行（レコード）毎に処理される
- 「#キーワード」のフィールドから始める

プログラムの基本構成

```
#include <stdio.h>
#include <math.h
int main(void) {
  double x, y;
  printf("Hello, world.\n");
  x = 0.1;
  y = sin(x);
  printf("SIN(%lf) = %lf\n", x, y);
  return 0;
}
```

- コンパイラで処理される「 」(ホワイトスペース)「;」「{」「}」「/*・・・*/」で区切られたバイト列

文とブロック

```
#include <stdio.h>
#include <math.h>
int main(void) {
  double x, y;
  printf("Hello, world.\n");
  x = 0.1;
  y = sin(x);
  printf("SIN(%lf) = %lf\n", x, y);
  return 0;
}
```

ブロック

文

プログラムの基本構成

```
#プリプロセッサ命令

#定数マクロ定義

int main(コマンドライン引き数) {

  変数の型 変数名; /*変数宣言*/

  実行すべき処理;

  return 0;

}
```


プログラミング言語の一般論

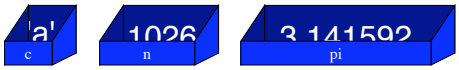
- ●データ型（定数と変数、 配列）
- ●代入・基本演算（四則演算）
- ●入出力
- ●条件分岐
- ●繰返処理
- ●関数、外部手続き

キーワード（予約語）

- データタイプ
  char int float double enum struct union typedef
- データタイプ修飾子
  signed unsigned short long const void
- データ記憶クラス
  auto static extern volatile register
- 制御構造
  if else do while for switch case break default return continue goto
- 演算子, 記号
  sizeof + - * / % & | , ; : [ ] ( ) { } #

変数

- データ（ビット列）を入れることができる箱
  - 名前がついている
  - 大きさが決まっている
  - 実行時に用意される（実行中ではない）



変数

- 文字型　[通常 8bit]
  char
- 整数型 [通常16bit or 32bit]
  int
- 拡大整数型 [通常32bit以上]
  long int, long
- 実数型(単精度浮動小数点数)[通常doubleと同一]
  float
- 倍精度実数型(倍精度浮動小数点数)[通常64bit]
  double

変数

Local Rule.1:

- 変数は全て32bit符号付き整数（int or long int）と64bit実数（double）を使う
  - 32bit符号付き整数
    −2147483648～2147483647
  - 64bit実数

変数宣言

データ型　変数名；

データ型　変数名，変数名，．．．
；

データ型　変数名＝初期値;

整数型	倍精度実数型
`int a;`	`double x;`
`int a, b;`	`double x, y, z;`
`int n=10;`	`double pi = 3.141592;`

定数

●整数（拡大整数）

`0, -1, 1234, (123456789L)`

- 倍精度実数型(倍精度浮動小数点数)

`0.0, 1., 123.4, 1.234e-3`

$1.234e\text{-}3 = 1.234 \times 10^{-3} = 0.001234$

- 文字列定数（文字列リテラル）

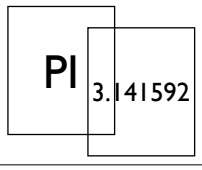
`"Hello world\n"`

記号定数 (マクロ置換)

- 表に名前、裏に値（定数）の書かれたカード
  - プリプロセッサの機能を利用する
  - 厳密な意味ではユーザ定義名前付きオブジェクトではない

#define 記号名 置換すべきテキスト

```
#define  N  1000
#define  PI  3.141592
double pi2;
pi2 = 2.*PI;
```



四則演算と代入

変数 ＝ 変数(定数) 演算子 変数(定

和	$y = a + b$	y = a + b
差	$y = a - b$	y = a - b
積	$y = a \times b$	y = a * b
商	$y = a \div b$	y = a / b
剰余	$y = a \bmod b$	y = a % b

四則演算と代入

整数 = 整数 + 整数; 実数 = 実数 + 実数;
整数 = 整数 - 整数; 実数 = 実数 - 実数;
整数 = 整数 * 整数; 実数 = 実数 * 実数;
整数 = 整数 / 整数; 実数 = 実数 / 実数;
整数 = 整数 % 整数;

Local Rule.2:

- 代入文の右辺と左辺の型を明示的に一致させる
- 自動的な型変換を用いず，常に明示的にキャストを行う

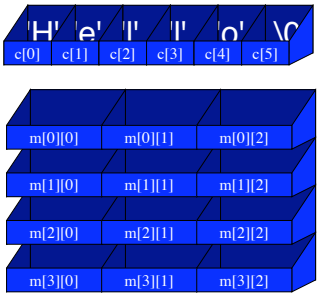
```
int n;
double a, x;
x = a / n;
```

```
int n;
double a, x;
x = a / (double) n;
```


配列

●配列名と数字(添字)で指定される変数の集まり

```
double a[100];
double b[10][5];

a[0] = 1.0;
a[1] = 2.0;
a[99] = 5.0;

b[0][0] = 1.2;
b[9][4] = 0.0;
```



一次元配列とベクトル

```
double b[3];
b[0] = 1.0;
b[1] = 2.0;
b[2] = 1.0;

double b[3]={1.0, 2.0, 3.0};

double b[3]=
  {1.0,
   2.0,
   3.0};
```

$$\vec{B} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

一次元配列と数列

```
#define N 100

int n;
double a, r, c[N];
c[n] = a*pow(r, n-1);

c[0] = a;
c[n] = r*c[n-1];
```

$$C_n = ar^{n-1}$$

$$\begin{cases} C_0 = 0 \\ C_n = rC_{n-1} \end{cases}$$

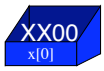
二次元配列と行列

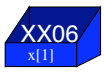
```
double a[2][2];
a[0][0] =  1.0;
a[0][1] = -1.0;
a[1][0] = -1.0;
a[1][1] =  2.0;
double a[2][2]={{1.0, -1.0}, {-1.0, 2.0}};
double a[2][2]=
     {{1.0, -1.0},
      {-1.0, 2.0} };
```

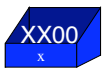
$$A = \begin{pmatrix} 1 & -1 \\ -1 & 2 \end{pmatrix}$$

配列に関する注意

```
int x[2][3] = { {0, 1, 2}, {4, 8, 16}} ;
```







Local rule:

- 配列の要素のアドレスを直接意識するようなプログラムにはしない。
- 必ずすべての（二次元なら2つ、三次元なら3つ）の添え字を指定する。

入出力

- 入力関数

```
getchar, getc, fgetc     ; 一文字入力
gets, fgets, fread       ; 文字列(データ列)入力
scanf, fscanf, sscanf    ; 書式付入力
```

- 出力関数

```
putchar, putc, fputc     ; 一文字出力
puts, fputs, fwrite      ; 文字列(データ列)出力
printf, fprintf, sprintf; 書式付出力
```

Local Rule.3:

- 入力は行わず，データはプログラム中に直接書く．
- 出力はprintfのみ．

printf 関数

```
int printf (const char *, 変数, 変数,...);
```

書式制御文字列リテラル

- %\"以外はそのまま出力
- "%+数字+文字"
  - %d　整数を出力
  - %f　固定小数点実数を出力
  - %e　浮動小数点実数を出力
- "%%" %自体を出力

その他
%i
%o
%u, %c, %s
%x, %X
%E, %g, %G

書式制御文字列

```
int n = 578;
```

	1	2	3	4	5	6	7	8	9	10
printf("%d\n", n);	5	7	8							
printf("%8d\n", n);						5	7	8		
printf("%08d\n", n);	0	0	0	0	0	5	7	8		
printf("%2d\n", n);	5	7	8							

書式制御文字列

```
double x = -57.89;
```

	1	2	3	4	5	6	7	8	9	10
printf("%f\n", x);	-	5	7	.	8	9	0	0	0	0
printf("%9.3\n", x);			-	5	7	.	8	9	0	
printf("%6.3d\n", x);	-	5	7	.	8	9	0			
printf("%6.1d\n", x);		-	5	7	.	9				

書式制御文字列

```
double x = -57.89;
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
printf("%e\n", x);	-	5	.	7	8	9	0	0	0	e	+	0	1		
printf("%15.7e\n", x);		-	5	.	7	8	9	0	0	0	0	e	+	0	1
printf("%10.1e\n", x);			-	5	.	8	e	+	0	1					

```
printf("%27.20e\n", x);
```

```
                5.78900000000000005684e+01
```

printf 関数

```
int printf (const char *, 変数, 変数,...);
```

書式制御文字列リテラル

Local Rule.4

- 原則

  %d　　整数

  %f　　実数

  %15.7e　実数

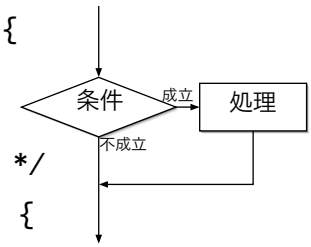
  を使用する

c言語の制御構造

- 単純if文
- if-else文 (ブロックif文)
- else-if文
- switch-case文
- while文
- do-while文
- for文

単純(ブロック)if文

```
if (条件式) {
  処理;
}
/* 例 a=|a| */
if (a < 0.) {
  a = -a;
}
```



条件式

●数値の大小関係の比較

$a > b$	a > b
$a \geq b$	a >= b
$a < b$	a < b
$a \leq b$	a <= b
$n = m$	n == m
$n \neq m$	n != m

Rule: 整数にしか使わない！

条件式

論理和	$T1 \oplus T2, T1 \vee T2$	(T1) || (T2)
論理積	$T1 \otimes T2, T1 \wedge T2$	(T1) && (T2)
論理否定	$\neg T$	! (T)

●論理演算

条件式 ≠ 0 ⇒ True (真)

条件式 = 0 ⇒ False (偽)

Local Rule:

- 条件式の値は使用しない

if文

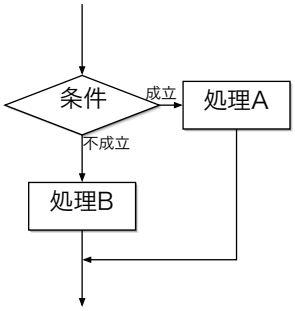
●単純if文の例

```
double b, d;
if (d >= 0.) b = -d;

int i, j, n;
if ((i - 1) < 0) j = i -1 + n;

int k, l; double d, eps;
if ((k == l) && (d < eps)) return 0;
```

if-else文

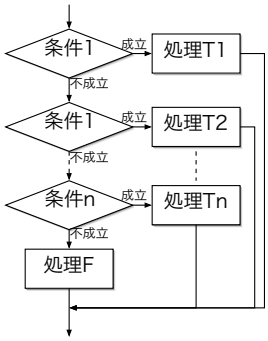
●ブロックif文

```
/* 例 a=|a| */
if (a >= 0.) {
  a = a;
}
else {
  a = -a;
}
```



else-if文

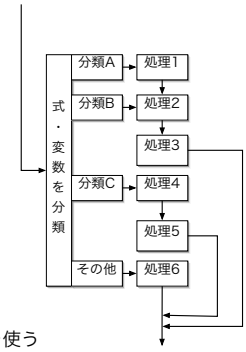
```
if (条件式1) {
  処理T1;
}
else if (条件式2) {
  処理T2;
}
.....
else if (条件式n) {
  処理Tn;
}
else{
  処理F;
}
```



switch-case文

```
swich (式) {
  case A: 処理1;
  case B: 処理2;
    処理3;
    break;
  case C: 処理4;
    処理5;
    break;
  default: 処理6;
    break;
}
```

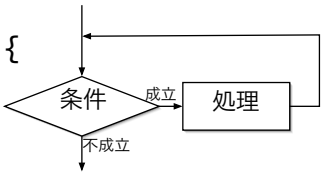
- Local Rule: else-ifを使う



while文

●条件前置型ループ

```
while (条件式) {
  処理;
}
```

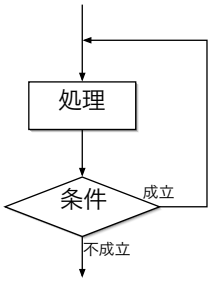


- Local Rule:
  無限ループを避けるためfor文を使用

do-while文

●条件後置型ループ

```
do {
  処理;
} while (条件式);
```

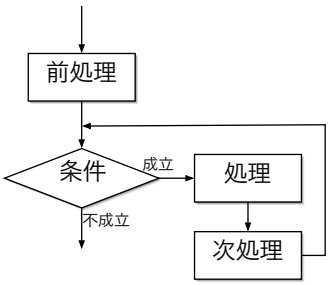


- Local Rule:
  無限ループを避けるためfor文を使用

for文

```
for(前処理; 条件式; 次処理){
  処理;
}
```

```
前処理;
while (条件式){
  処理;
  次処理;
}
```



forループ

●1+2+3+4+...+100を計算する

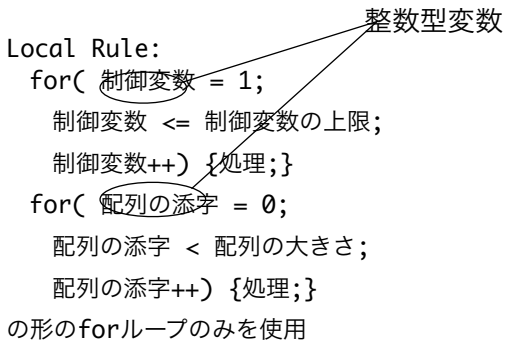
```
#define N 100
int i, s;
s=0;
for (i=1; i<=N; i=i+1) s = s + i;
```

Local rule:

- --, +=, /=, *=等は使わない
- ++のみ使う

~~for (i=1, s=0; i<=N; i++) s+=i;~~

forループ



の形のforループのみを使用

forループ

使用例

- 特定の処理をＮ回繰り返す

```
#define N 100
int i;

for (i=1; i<=N; i++){
  printf("Loop %6d\n", i);
}
```


forループ

使用例

- 配列各要素について処理を行う

```
#define N 100
int i; double a[N];

for (i=0; i<N; i++){
  printf("A(%3d)=%15.7f\n", i, a[i]);
}
```


forループ使用上の注意

- 制御変数増加型のループのみ使用
- 制御変数をループ内で変更しない
- 制御変数は整数型を用いる
- 初期値は０か１
- 増分は１のみ
- 処理は必ずブロックにする

forループの例

● 配列の値を初期化する

```
#define N 10

int i;
double a[N];

for (i=0; i<N; i++){
  a[i]=1.0;
}
```

forループの例

● 二次元配列に単位行列を代入する

```
int i, j;
double a[N][N];
for (j=0; j<N; j++){
  for (i=0; i<N; i++){
    if (i==j) { a[i][j]=1.0;}
    else      { a[i][j]=0.0;}
  }
}
```


forループの例

● 配列に数列を代入する

```
int i;
double a, r, c[N];
for (i=0; i<N; i++){
  if (i==0) {
    c[i] = a;
  }
  else {
    c[i] = r*c[i-1];
  }
}
```

$$c_n = ar^n$$
$$\begin{cases} c_0 = a \\ c_n = rc_{n-1} \end{cases}$$

forループの例

- 配列の総和を求める

```
int i;
double s, c[N], *p;

s = 0.0;
for (i=0; i<N; i++) {
  s = s + c[i];
}
```

$$S = \sum_{i=0}^{N-1} c_i$$

~~for (p=c, i=0, s=0.0; i<N; i++, s+=*p++);~~

forループの例

● 配列の最大要素を求める

```
int i, imax;
double vmax, val[N];
imax = 0;
vmax = val[0];
for (i=1; i<N; i++){
  if (val[i]>vmax) {
    imax = i;
    vmax = val[i];
  }
}
```


forループの例

$\vec{a} = (a_0, a_1, \cdots, a_{n-1})$
$\vec{b} = (b_0, b_1, \cdots, b_{n-1})$
$\vec{a} \cdot \vec{b} = a_0b_0 + a_0b_0 + \cdots a_{n-1}b_{n-1} = \sum_{i=0}^{n-1} a_ib_i$

● ベクトルの内積の計算

```
int i;
double c, a[N], b[N];
c = 0.0;
for (i=0; i<N; i++){
  c = c + a[i]*b[i];
}
```


forループの例

● 行列の積の計算

```
int i, j, k;
double a[N][N], b[N][N], c[N][N];
for (i=0; i<N; i++){
  for (j=0; j<N; j++){
    c[i][j] = 0.0;
    for (k=0; k<N; k++){
      c[i][j] = c[i][j] + a[i][k]*b[k][j];
    }
  }
}
```

$$C_{ij} = \sum_{k=0}^{N-1} A_{ik}B_{kj}$$

forループの例

- 周期境界条件を用いる

```
int j, k;
double a[N], b[N], c[N];
for (k=0; k<N; k++){
  c[k] = 0.0;
  for (j=0; j<N; j++){
    c[k] = c[k] + a[j]*b[j+k];
  }
}
```

$$c_k = \sum_{j=0}^{N-1} a_j b_{j+k}$$

$$b_{k+N} = b_k$$

forループの例

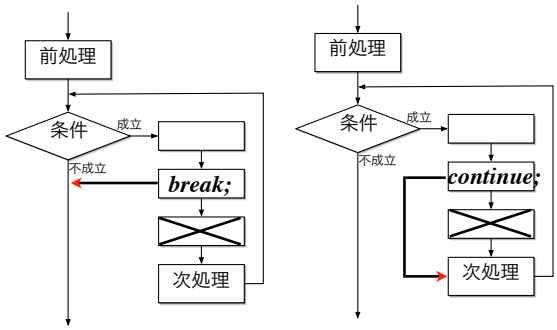
- 周期境界条件を用いる

```
int j, k;
double a[N], b[N], c[N];
for (k=0; k<N; k++){
  c[k] = 0.0;
  for (j=0; j<N; j++){
    jp = j + k;
    if (jp >= N) jp = jp - N;
    c[k] = c[k] + a[j]*b[jp];
  }
}
```

$$c_k = \sum_{j=0}^{N-1} a_j b_{j+k}$$

$$b_{k+N} = b_k$$

breakとcontinue



関数

●標準ライブラリ関数(ANSI/POSIX)

- 環境依存ライブラリ関数(API)
- ユーザ定義関数
  - 値を返す（数学関数など）
  - 値を返さない（サブルーチン）

標準数学関数

- math.h に定義されている
- 使用するときには #include <math.h>

注意：全て，引数も返り値もdouble型

平方根	$\sqrt{x}$	sqrt(x)	ただし、$x \geq 0$
指数関数	e^x	exp(x)	
自然対数	$\ln x, \log_e x$	log(x)	ただし、 $x > 0$
常用対数	$\log_{10} x$	log10(x)	ただし、 $x > 0$
絶対値	$\lvert x \rvert$	fabs(x)	

標準数学関数

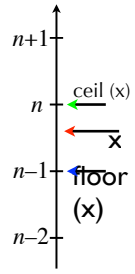
●三角関数・逆三角関数

正弦	$\sin x$	sin(x)	
余弦	$\cos x$	cos(x)	
正接	$\tan x$	tan(x)	戻り値をzとおくと、
逆正弦	$\arcsin x$	asin(x)	$\lvert x \rvert \leq 1, -\pi/2 \leq z \leq \pi/2$
逆余弦	$\arccos x$	acos(x)	$\lvert x \rvert \leq 1, 0 \leq z \leq \pi$
逆正接	$\arctan x$	atan(x)	$-\pi/2 \leq z \leq \pi/2$
	$\arctan(x / y)$	atan2(x, y)	$-\pi \leq z \leq \pi$

標準数学関数

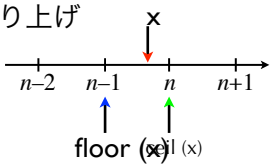
- 双曲線関数

$\sinh x = (e^{x} - e^{-x})/2$	sinh(x)
$\cosh x = (e^{x} + e^{-x})/2$	cosh(x)
$\tanh x = \sinh x/\cosh x$	tanh(x)



- 切り捨て・切り上げ

floor (x)
ceil (x)



ユーザ定義関数

- プロトタイプ宣言
  戻り値の型 関数の名前(引数の型, 引数の型, ...);
- 関数の呼び出し
  変数 = 関数の名前(引数, 引数, ...);
- 関数の定義
  戻り値の型 関数の名前(引数の型 仮引数, ...) {
    処理;
    return 戻り値;
  }

値を返さない関数

- プロトタイプ宣言
  void 関数の名前(引数の型, 引数の型, ...);
- 関数の呼び出し
  関数の名前(引数, 引数, ...);
- 関数の定義
  void 関数の名前(引数の型 仮引数, ...) {
    処理;
  }
  FORTRANやPascalのサブルーチンに相当する

関数の例

- 単純な初等関数の組み合わせ
  $f(x) = e^{-x} - 1$

```
double fnct(double);

double fnct(double x){
  double f;
  f = exp(-x) - 1.;
  return f;
}
```


関数の例

- 単純な初等関数の組み合わせ（2変数）
  $f(x, y) = x^6 - y^3$

```
double fnct(double, double);

double fnct(double x, double y){
  double x3, x6, y3, f;
  x3 = x*x*x;
  y3 = y*y*y;
  x6 = x3*x3;
  return (x6 - y3);
}
```


関数の例

- 配列の要素の最大値

```
double maxelement(double a[], int n) {
  int i; double maxval;
  maxval = a[0];
  for (i=1; i<n; i++) {
    if (a[i] > maxval) maxval = a[i];
  }
  return maxval;
}
```

def_const.h

```
#define N 3
int last(int*);
```

main.c

```
#define N 3
int last(int*);
int main(void){
   int b,a[N]={1,2,3};
   b=last(a);
   printf("%d\n",b);
   return 0;
}
```

function.c

```
#define N 3
int last(int c[]){
  return c[N-1];
}
```

main.c

```
#include "def_const.h"
int main(void){
   int b,a[N]={1,2,3};
   b=last(a);
   printf("%d\n",b);
   return 0;
}
```

function.c

```
#include "def_const.h"
int last(int c[]){
  return c[N-1];
}
```


main.c

```
#define N 3
int last(int*);
int main(void){
   int b,a[N]={1,2,3};
   b=last(a);
   printf("%d\n",b);
   return 0;
}
```

function.c

```
#define N 3
int last(int c[]){
  return c[N-1];
}
```

main.c

```
#define N 3
int last(int*, int);
int main(void){
   int b,a[N]={1,2,3};
   b=last(a, N);
   printf("%d\n",b);
   return 0;
}
```

function.c

```
int last(int c[],int m){
  return c[m-1];
}
```


関数と配列

```
#define N 2
#define M 3
int last(int c[][M]);
int main(void){
  int b, a[N][M] = {{11, 12, 13},
                    {21, 22, 23}};
  b = last(a);
  printf(”%d\n”, b);
  return 0;
}
int last(int c[][M]){
  return c[N-1][M-1];
}
```

```
a[0][3] = {11, 12, 13};
a[1][3] = {21, 22, 23};
a[2] = {a[0],
          a[1]}
```


関数と配列

Rocal Rule:

- 1次元配列の大きさは明示するか関数に引数として渡す
- 2次元配列の大きさは配列初期化や関数宣言，プロトタイプ宣言でも明示する

関数と引数

```
int main(void){
  int a   =  1 ;
  func(a);
  printf("%d\n",a   );
  return 0;
}
void func(int c  ){
  printf("%d\n",c   );
  c    = c    + 1;
}
```

```
% a.out
1
1
%
```


関数と引数

```
int main(void){
  int a[] = {1};
  func(a);
  printf("%d\n",a[0]);
  return 0;
}
void func(int c[]){
  printf("%d\n",c[0]);
  c[0] = c[0] + 1;
}
```

```
% a.out
1
2
%
```

関数の例

- 行列の積を計算する

```
void mulmat(double a[N][N],double b[N][N],
            double c[N][N]){
  int i, j, k;
  for (i=0; i<N; i++) { /* l3 */
    for (j=0; j<N; j++) { /* l2 */
      c[i][j] = 0.;
      for (k=0; k<N; k++) { /* l1 */
        c[i][j] = c[i][j] + a[i][k]*b[k][j];
      } /* l1 */
    } /* l2 */
  } /* l3 */
}
```


関数と引数

- 呼び出された側(callee)の引数への処理は,
  呼び出した側(caller)には反映されない
  (call by value)
- 引数が配列名のときには,
  呼び出された側(callee)の引数への処理が,
  呼び出した側(caller)に反映される
  (疑似的call by reference)