配列

配列

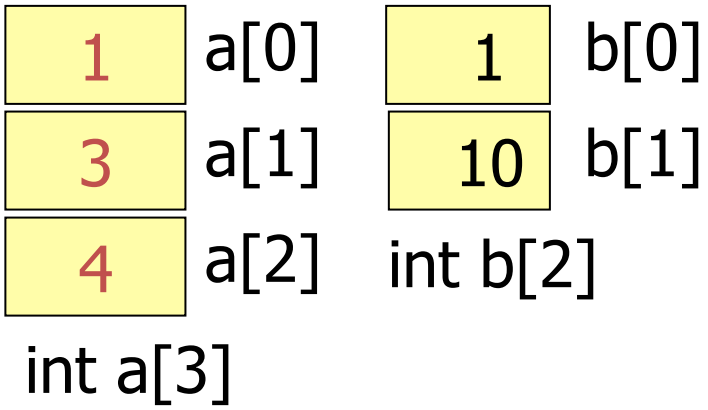
- 処理の繰り返し
  - 複数の同種類の変数を同じ名前で一括して扱う
  - 変数の並び → 配列
  - 何番目か？ → 添字（最初は0で始まる）
    - 添字は数字なので式が使える

宣言：

```
int a[3];
int b[2] = {1, 10};
```

利用例：

```
a[0] = 1; a[1] = a[0] + 2;
a[2] = a[0] + a[1];
```



- フィボナッチ数列 $f_1 = f_2 = 1$, $f_{n+2} = f_{n+1} + f_n$,の計算(f_{10}を求める)

```
int i, f[10];
    f[0] = f[1] = 1; /* f_n → f[n-1] となる */
    for (i = 0; i < 8; i++) {
          f[i + 2] = f[i + 1] + f[i];
    }
```

- キーボードから連続データ入力

```
int i, x, v[10];
for (i = 0; i < 10; i++) {
    scanf (“%d”, &x);
    v[i] = x;
}
```

補足：データ入力関数 scanf

- キーボード入力を読み込む
- scanf のところに達したら、入力待ちになる
- 読み込んだ値を、そのまま返す
- printf と同様に，読み込むデータ型を指定

```
int a;
scanf(“%d”, &a); /* &をつける理由は後日説明 */
```

補足：繰り返し読み込みを手動で終了させる

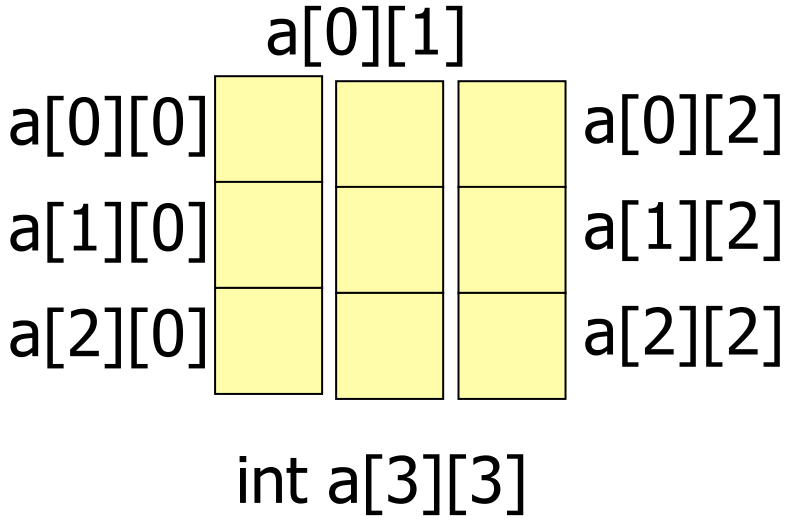
```
int x, sum;
sum = 0;
while (scanf (“%d”, &x) != EOF) {
   sum += x;
}
```

ファイルの終わり(EOF)はCtrl-Dでキー入力

２次元の配列

- 変数の並び：２次元的な配置
- 添え字：２個の正数を用いる（行列表記）

```
int a[3][3];
a[0][0] = 1;
a[1][2] = a[2][0] + a[0][2];
```



２次元配列の処理

- 多重化された繰り返し文を用いる

```
int m[10][5], mt[5][10], i, j;
for (i = 0; i < 10; i++) {
   for (j = 0; j < 5; j++) {
    /* 転置行列 mt の計算 */
    mt[j][i] = m[i][j];
   }
}
```

配列を関数へ渡す方法

```
int main () {
    int a[10], s;
    /* データの読み込み */
    s = calcSum(a);
    /* 出力等 */
}
```

配列の名前 a で渡す
a はアドレスのデータとなる

```
int calcSum (int data[]) {
    int i, sum=0;
    for (i = 0; i < 10; i++)
      sum += data[i];
    return sum;
}
```

受け取る関数側では [] が必要！

二重配列の渡し方

```
int main () {
    int mat[3][3], s;
    /* データの読み込み */
    s = calcSum(mat);
    /* 出力等 */
}
```

配列の名前 mat で渡す

```
int calcSum (int m[][3]) {
    int i, j, sum=0;
    for (i = 0; i < 3; i++)
      for (j = 0; j < 3; j++)
        sum += m[i][j];
    return sum;
}
```

受け取る関数側では [][] が必要
2番目の[]内には配列数を記入！

文字列とファイル入出力

文字

- char 型の変数に入っている
  - 単なる数字
- 数字と文字の対応が決まっている
  - 文字コード(通常、英数字はASCIIコード)
  - 例) 65なら'A'
- ターミナルでのコード表の見方

  % man ascii

文字の種別判定

- 数字の判定

  if (‘0’ <= c && c <= ‘9’)

- 英字の判定

  if (‘a’ <= c && c <= ‘z’ ||

  ‘A’ <= c && c <= ‘Z’)

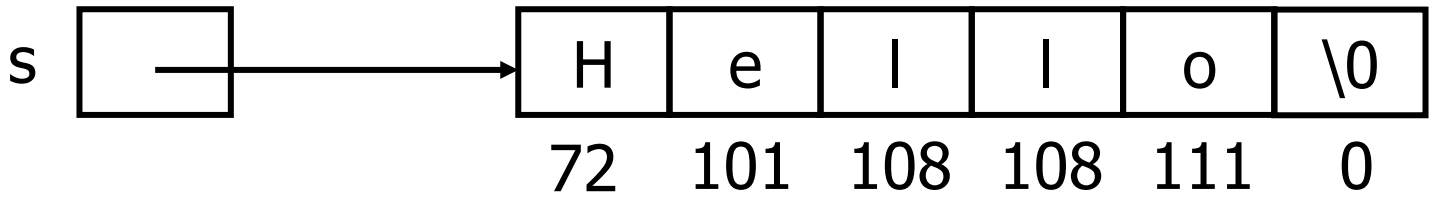
- 空白文字の判定

  if (c == ‘ ‘ || c == ‘\n’ || c == ‘\t’)

文字列

- C言語には文字列をあらわす型はない
  - string型のような文字列の型をもつ言語もある
- 文字の配列を文字列として扱う
  - 配列の先頭アドレスで文字列を参照する
- 文字列は必ずヌル文字 '\0' で終わる

char *s = "Hello"; s は先頭アドレスの値



文字列(注意事項）

- “Hello”のようにプログラム中に書いた文字列は書き替えることができない(文字列定数)
- 配列を取るときには、文字数より一文字分多くしなければならない
  - ヌル文字の分

文字列操作その１（文字数の計算）

```
char str[100];
int n = 0; /* n は文字位置を先頭から進める */
scanf (“%s”, str); /* キーボードからの読み込み */
while (str[n] != ‘\0’)
    ++n; /* 文字位置を一つ進める */

printf(“文字数 = %d”, n); /* str[n] が ‘\0’ なので */
```

文字列操作その２（文字列比較）

```
char str[100], *tut = “Toyohashi Unv.”;
int n = 0; /* n は文字位置を先頭から進める */
scanf (“%s”, str); /* キーボードからの読み込み */
while (str[n] != ‘\0’ && tut[n] != ‘\0’) {
    if (str[n] != tut[n])
       return 0; /* １文字でも違えば0を返す */
    ++n;
}
if (str[n] == ‘\0’ && tut[n] == ‘\0’)
    return 1; /* 文字数が最後まで一致 */
else return 0; /* 一方の文字列が尻切れ */
```

補足：キーボードからの文字列入力

- scanfを使う場合

```
char str[100];
scanf("%s", str); /* &str ではなく，str を与える */
```

- 入力の終わりの判定
  - キーボードから ^D (コントロールD)を入力すると、入力終わり(ファイルの終わり)になる
  - scanfは入力終わりに当たるとEOFを返す

```
while(scanf("%s", str) != EOF) {
  …
}
```

- ファイルにデータを入力しておいて

```
% cmd < test.txt
```

ファイルへの読み書き

```
FILE *fptr; /* ファイルへのポインタ */
char str[100];
/* ファイルを開く （読み込みモード）*/
if ((fprt = fopen (“test.txt”, “r”)) != NULL) {
    /* ファイルからの読み込み */
    while (fscanf (fptr, “%s”, str) != EOF) {
        printf (“%s”, str); /* 読み込みの出力 */
    }
    /* ファイルを閉じる */
    fclose (fptr);
}
```

ライブラリ関数に必要なヘッダファイル

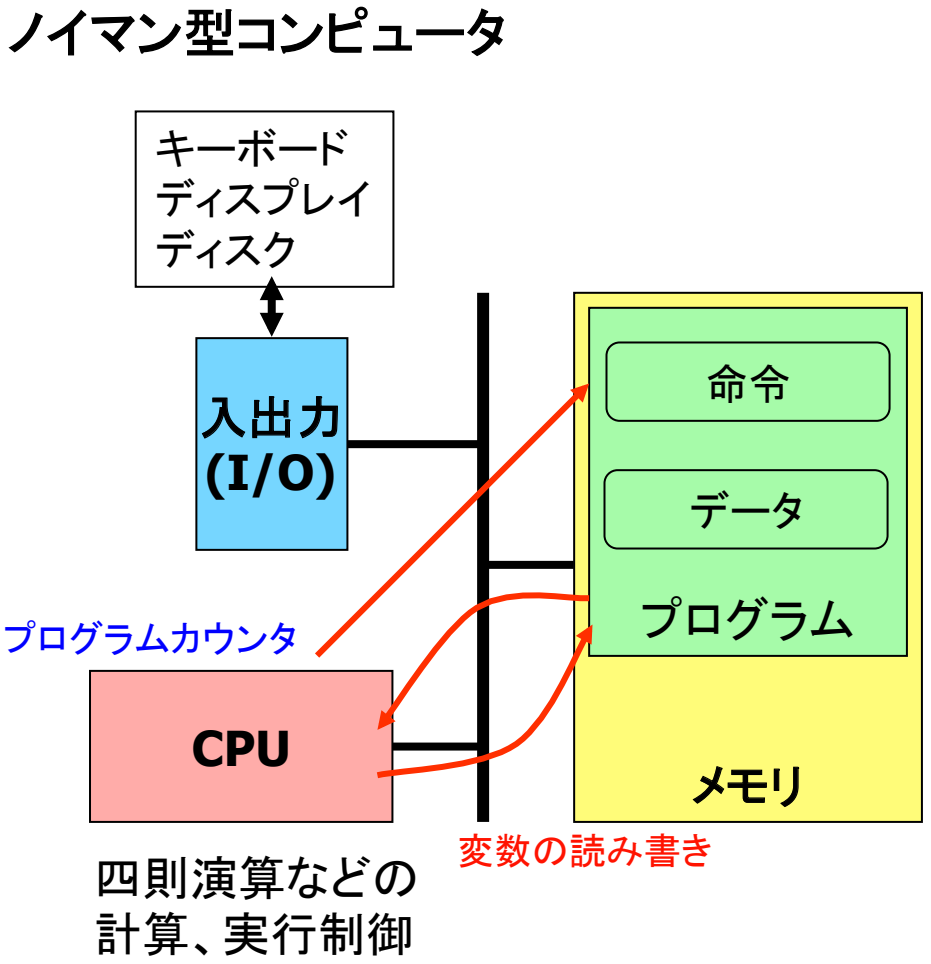
- stdio.h : 入出力に関わる関数群
- string.h : 文字列操作に関わる関数群
- math.h : 数学関数(sin, cos, tan, etc.)
- ファイルの先頭で、ヘッダファイルを含める必要

  #include <stdio.h>
- これらのファイルは決められたフォルダに置かれている
- < > 括弧は、そのフォルダの位置を暗黙に指定している
- ユーザが作成したライブラリーには “” 括弧を用いる

再帰呼び出し

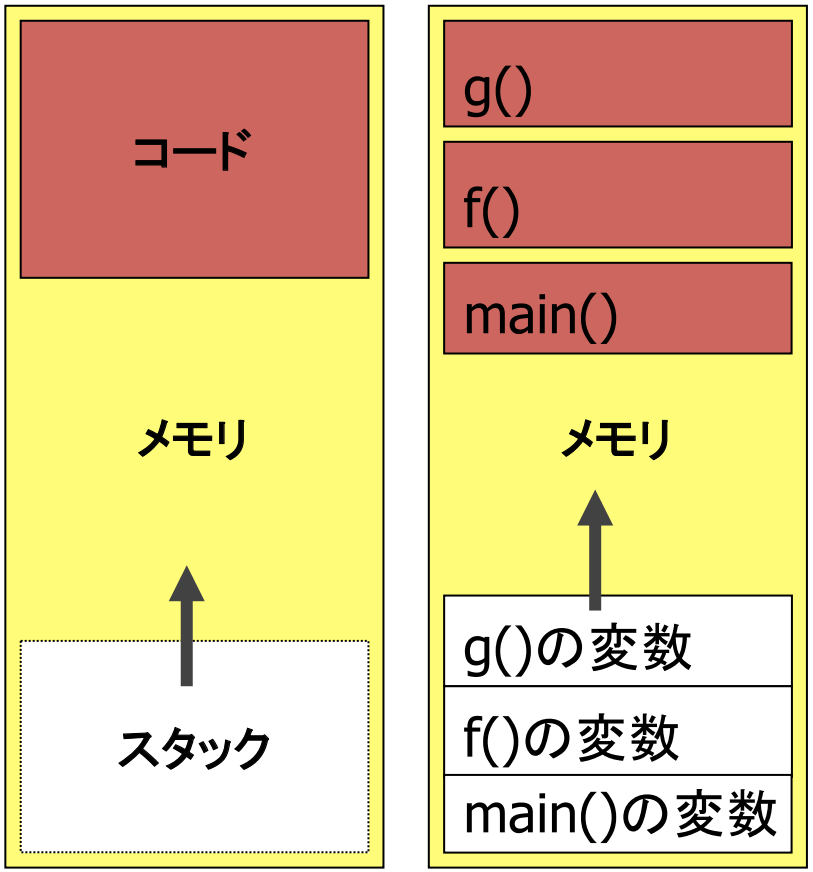
コンピュータの実際（復習）

- コンピュータはどうやって動いている？
  - 命令もデータも同じメモリ上にある
  - 命令を一つずつ進め、命令に応じてデータの部分にアクセス



関数呼び出しの実際(復習)

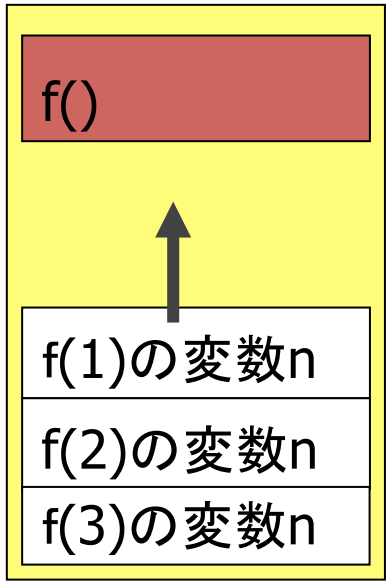
- 関数中でのみ有効なデータの領域
  - 関数が**呼び出されるごとに確保**される
    - 実行時の環境
  - スタック領域
  - コード領域



再帰呼び出し

- 自分で自分を呼んだらどうなるか
  - 実行領域は各呼び出しごとに確保される
  - あたかも違う関数が呼ばれたみたいに動く
  - 呼び出しが終わると元に戻る
- forやwhileとは異なる形の繰り返し
- 再帰呼び出しという
  - recursion call ⇔ iteration call

```
void f (int n) {
        if (n > 0) f (n-1);
        else return;
}
```



再帰呼び出し(実装例)

```
int f(int x) {
        pinrtf("f(%d) called.\n", x);
        if (x == 0)
                return 1;
        else
                return(f(x-1));
}
```

```
f(5)
f(5) called.    f(4)を呼ぶ
f(4) called.    f(3)を呼ぶ
f(3) called.    f(2)を呼ぶ
f(2) called.    f(1)を呼ぶ
f(1) called.    f(0)を呼ぶ
f(0) called.    return 1
```

再帰呼び出しの例

- 関数はどう呼ばれるのか
  - 終了条件は必須！

```
int fact(int x) {
        if (x == 1)
                return 1;
        else
                return(x * fact(x-1));
}
```

```
fact(5)
=5 * fact(4)
=5 * 4 * fact(3)
=5 * 4 * 3 * fact(2)
=5 * 4 * 3 * 2 * fact(1)
=5 * 4 * 3 * 2 * 1
```

例題：最大公約数の計算

- x と y が整数で、少なくともどちらか一方が0でない場合、両方を割り切る最大の整数が最大公約数gcd (x, y)

[例： gcd(100, 60)=20, gcd(-6, 9) = 3, gcd(0, 12) = 12]

- もし、ある正数が2つの正数 x と y を 割り切るならば、同様にして2つの整数 (x - y) と y を 割り切ることができ、その逆も成り立つ。以上の性質から、任意の x, y の組に対して、gcd(x, y)=gcd(y, x), gcd(x, y)=gcd(x-y, y) となる
- もし x > y であれば x - y は x より小さく、gcd(x, y) を gcd(x-y, y) で置き換えることは簡単化を意味する。また、明らかに gcd (x, 0) = x である

例題：最大公約数の計算（要約）

- ユークリッドの互除法: x と y の最大公約数をgcd(x, y)で表すと、以下が成立する
  - gcd(x, y) = gcd(y, x) : 置換
  - gcd(x, y)=gcd(x-y, y) : 簡単化[if (x > y)]
  - gcd(x, 0)=x : 最終解
- 置換と簡単化の手続きを、最終解の状態に到るまで繰り返す

例題：最大公約数の計算

- 計算実行例：

gcd(700,500) = gcd(200,500)=gcd(500,200)
=gcd(300,200)=gcd(100,200)=gcd(200,100)
=gcd(100,100)=gcd(0,100)=gcd(100,0)= **100**

gcd(13,3)=gcd(10,3)=gcd(7,3)=gcd(4,3)=gcd(1,3)
=gcd(3,1)=gcd(2,1)=gcd(1,1)=gcd(0,1)=gcd(1,0)= **1**

最大公約数の計算の実装例

```
int main (int argc, void *argv[]) {
  int n0, n1;
  sscanf (argv[1], "%d", &n0);
  sscanf (argv[2], "%d", &n1);
  printf ("%d と %d の最大公約数は %d です。",
          n0, n1, gcd (n0, n1));
}
int gcd (int n0, int n1) {
  if (n1 == 0)
    return n0;
  else if (n0 < n1)
    return gcd (n1, n0);
  else
    return gcd (n0 - n1, n1);
}
```