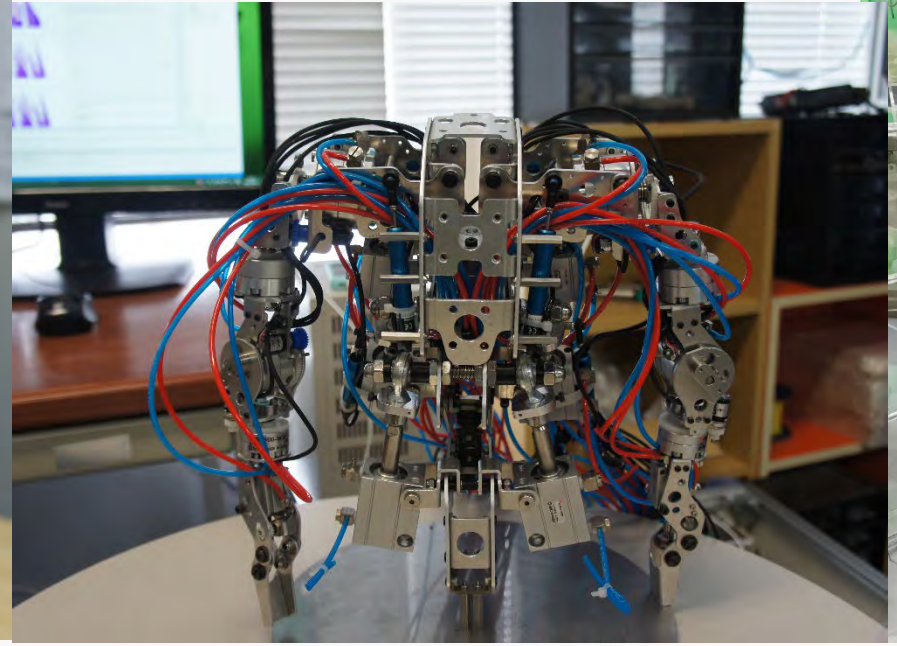




計算機とプログラミング

第２回　基本文法１

アンドロイドを高機能化するための
設計・評価・制御法の研究をしています
C，python，C++，C#，MATLAB，R
などを用途に応じて活用しています

担当：石原尚
機械工学専攻　動的システム制御学領域　講師

C言語の「使い方の基礎」を学びます

ここまでを使いこなしておくことが必要

第１回
基礎を学ぶ準備
- 授業の進め方
- C言語の立ち位置
- C言語の環境整備

杉本先生

第２～４回
使い方の基礎
- 変数と制御構造
- ＋関数
- ＋配列

今日はここ！

どう使うかをPythonとの違いも含めて解説します

石原

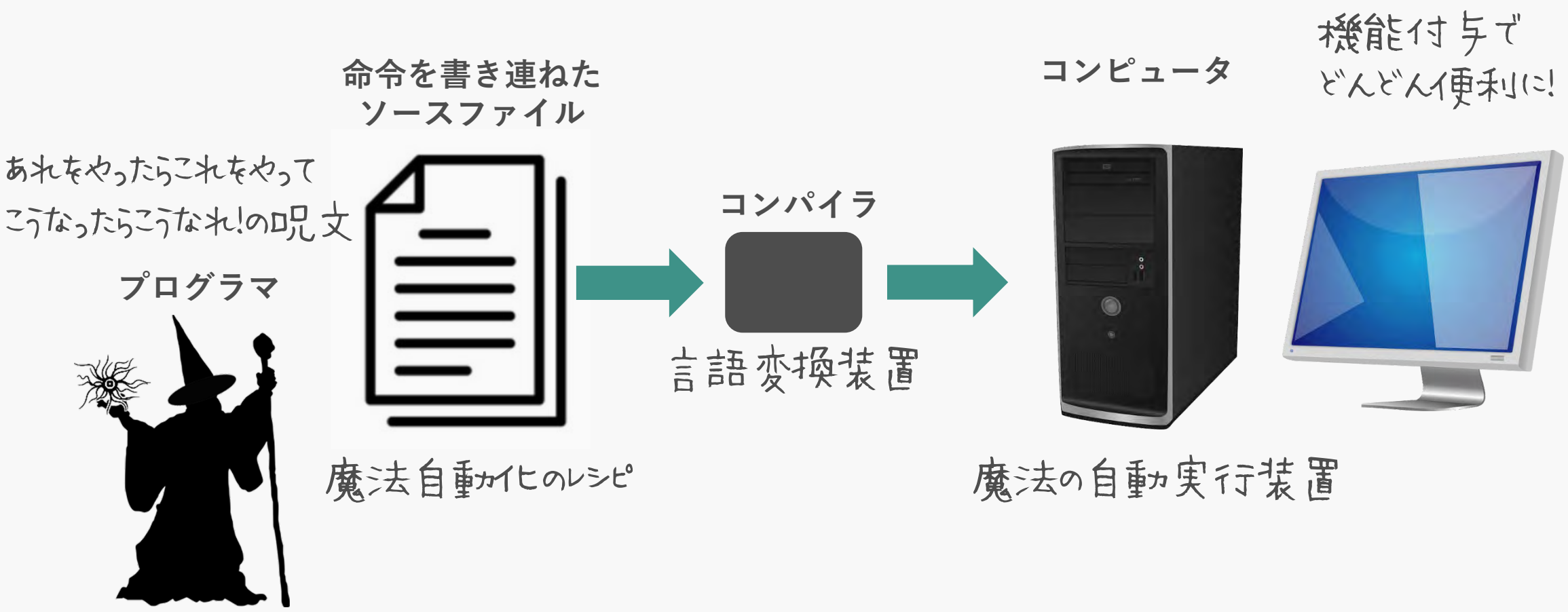
第５～８回以降
少し難しい使い方
- ポインタ
- データ構造

杉本先生＆平原先生

第９～１３回
応用と実践
- アルゴリズム
- 総合演習

中西先生＆杉本先生＆石川先生

C言語は人がコンピュータに機能を与える１つの手段



C言語は記述量が多い分だけ厳密に操れる

気軽に機能を作っていきたいならPython

プログラマ

計算機械を使いこなしたいならC

Pythonのソースファイル

ざっくりでお任せ!

簡潔なレシピ

細かいところはそれなりにやっておきます

コンパイラ

コンピュータ

Cのソースファイル

厳密なレシピ

コンパイラ

勝手にはできないので細かく指示してください!

コンピュータという計算機械を厳密に操れる!

Hello worldには基礎的な学習内容が詰まっている

Pythonの場合

```
print("Hello World!")
```

main関数は「実行命令の有効範囲を決める」特別な関数

基礎学習では
この形でしか使いません

int main()で覚える

前回扱ったソースファイル
lesson1.cの中身

```
#include <stdio.h>
int main( )
{
    printf("Hello World.¥n");
}
```

開始

終了

ここに書かれた
実行命令だけ有効

{ }の範囲はブロックと呼ばれる

範囲を決める括弧が大事

プログラムを読むときはまずmain関数の括弧の範囲を見ると良い

命令文は「定義・実行命令」を担う重要要素

定義と似た「宣言」もあるがここでは扱わない

前回扱ったソースファイル
lesson1.cの中身

```
#include <stdio.h>
int main( )
{
    printf("Hello World.¥n");
}
```

呪文発動!のイメージで
セミコロンを打つとよい

複雑な実行命令「以外」の
命令文の終わりには必ずセミコロン

命令文は
- **定義**
- **単純な実行命令**
- **複雑な実行命令**

の３種を知っておけば十分

Pythonでは改行さえすれば
セミコロンなしでOKでした.
C言語では改行は無視されるので,
長い命令も改行してOKです.

定義…変数の型の定義など後の実行の準備
単純な実行命令…printf関数など, ブロックを使わない命令
複雑な実行命令…whileなど, ブロックを使う命令

#include<>はよく使う命令文を簡単に使えるようにする

Pythonではimport

意味

**「stdio.hというファイル
の中身をここに埋め込みなさい」**

インクルードは，便利な魔法を使える使い魔を召喚する呪文ようなもの．
stdio.hは，データの読み書きが得意な使い魔．

**前回扱ったソースファイル
lesson1.cの中身**

```
#include <stdio.h>
int main( )
{
    printf("Hello World.¥n");
}
```

**stdio.hによってprintf()以外にも，
・キーボード入力を読む scanf()
・ファイルを開く fopen()
などが簡単に使えるようになる**

stdio.h以外にも，便利な使い魔がたくさんいる．

**冒頭の#は，main関数実行前に
処理されるべきことを示す
「プリプロセッサ指令」のマーク**

#をつけていないと実行されない

**stdio.hをincludeしていなくても
ややこしく書けば実現できないことはない．**

面倒なので，これやっといて，と使い魔に投げる感じ

バグを防ぎ，デバッグを容易にするために整理して書く

前回扱ったソースファイル
lesson1.cの中身

```
#include <stdio.h>
int main( )
{
    printf("Hello World.¥n");
}
```

余白部分に「全角スペース」が絶対入らないようにする

コンパイルエラーが出るのに
見えない!直しにくい!

ブロックの終わりを示す括弧と左右位置を揃える

揃えなくてもコンパイルは通るが
この方が有効範囲を確認しやすい

セミコロンをつけたら改行する

改行しなくてもコンパイルは通るが，
この方が命令の流れを追いやすい

ブロックの中の命令文はTabキーで右にずらしておく

ずれの量に関係なくコンパイルは通るが
この方がブロック内の構造を把握しやすい

基本的な構造を作っていく方法を「おさらい」します

C言語では，変数は「代入」する前に「定義」できる

変数定義 ＝ 変数に入る値を保存しておく記憶領域（メモリ）のサイズを指定し確保する命令文

C言語 - 有限の記憶領域を無駄なく管理できる

変数定義

この変数のために
このサイズのメモリを
確保せよ!の呪文

変数代入

メモリへの値の保存.
保存前には変な値が入っている可能性有

```
int main( )
 {
    int a;
    double b;

    a = 5;
    b = 3.2;
 }
```

```
int main( )
 {
    int a = 5;
    double b = 3.2;
}
```

変数定義&代入

これでも良い

Python – 記憶領域管理はお任せ

```
a = 5
b = 3.2
```

変数代入のみ

確保するメモリ
の大きさは
代入値に応じて
勝手に決まる
(どのサイズになって
いるかは調べないと
わからない)

もちろん他にも色々ある

C言語の変数の型は確保されるメモリサイズが固定

これ以上メモリが使われる恐れはない

1つのプログラムの暴走でコンピュータのメモリが圧迫される恐れ

格納できるデータ	C	Python
整数	**int** 4 byte	**integer** 桁数に応じて増加
実数	**double** 8 byte	**float** 8 byte
文字	**char** 1 byte（1 文字限定）	**string** 文字数に応じて増加

C言語のfloatは4byteなので別物

C言語では文字列を扱うのが少し難しい

基本的な演算子はCとPythonで共通

演算	C	Python
加算	+	+
減算	-	-
乗算	*	*
除算	/	/ or //
剰余	%	%
累乗	なし	**
インクリメント 値を１増やす	++	なし
デクリメント 値を１減らす	--	なし

Pythonの方がより柔軟だがほぼ同じように使える

C言語ではこれが便利

C言語では累乗計算面倒

int型の除算・計算順・型の値域に注意しないと計算ミス

int型の除算

小数以下は「切捨」

1÷3の計算でintを使うと…

```
int a = 1;
int b = 3;
double c = a/b;
```

整数になるように
切り捨てられてから
cに代入される

➡ cは0.000…

計算順

%*/ が +- に優先

これを忘れると想定外の値になる

```
int a = 3+5*2/3;
```

ここは左から

```
int a = 3+ 10 /3;
```

切り捨て!

```
int a = 3+  3;
int a =   6;
```

上記の流れを踏まえて,

```
int a = 3+5/3*2;
```

の式の答えを考えてみましょう.

型の値域

値域を出ると変な値に

知らずにこんな計算をすると…

```
int a = 1000000000;
int b = 4;
int c = a*b;
```

➡ cは4000000000
ならない！

オーバーフロー

int型では

最大値	2147483647
最小値	-2147483648

これらに気をつければ多少はミスが減ります

int型の除算

小数重要ならdouble

はじめからdoubleにしておく

```
double a = 1.0;
double b = 3.0;
double c =  a/b;
```

もしくは直前で定義変更

```
int a = 1;
int b = 3;
double c =
    (double)a/(double)b;
```

型のキャスティング

計算順

()で優先度調整

()をつけると最優先で計算される

```
int a = 3+5/3*2;
```

➡ cは5

```
int a = 3+5/(3*2);
```

➡ cは3

不安なときはカッコをつける!

型の値域

除算を先に済ませる

```
int a =
40000*100000/4000;
```

➡ aが変な値に

左の乗算でオーバーフロー

```
int a =
40000/4000*100000;
```

➡ aは正しい値に

左の除算で小さい値に

printf()で色々できる

Pythonではprint()

文字列だけの表示

```
printf(“Hello ”);
printf(“World”);
```

➡ Hello World

pythonとは違って
勝手には改行されない

特別な記号

“ ” の間に差し込める

```
printf(“He¥tllo¥nWorld”);
```

¥tでタブスペース

➡ He　　llo
World

¥nで改行

文字列中への変数埋め込み

整数は%d　実数は%f
文字は%c　文字列は%s
で**各型の変数埋込位置を指定**

中身は追って伝えるのでまってて,みたいな指示

```
char a = ‘r’;
int b = 3;
printf(“%cは%d”, a,b);
```

➡「rは3」が表示

指示した順で追記

Pythonではinput()

scanf()でキーボード入力を変数に代入できる

基本の読み込み（１つの整数）

代入する変数は先に定義しておかないとダメ

```
int a;
scanf(“%d”,&a);
```

実行後，コンソールに整数を入力してEnterキーを押すと変数aにその整数が入る

1つの整数を読み込め，という指示　　aに代入せよ，という指示．変数名の前には「&」をつける!

少し難しい読み込み（実数と文字の読み込み）

```
double a;
char b;
scanf(“%lf %c”,&a,&b);
```

「5.05 y」のように半角スペースを挟んで実数と文字を入力してEnterキーを押すとそれぞれがaとcに代入される

1つの実数の後で1つの文字を読み込め，という指示　　aに代入できたら，次にbに代入せよ，という指示

複数段階の分岐にも対応

条件分岐をさせたいときはif

ifを使った制御構造のテンプレート

```
if(条件式①){
    命令文①;
}else if(条件式②){
    命令文②;
}else{
    命令文③;
}
```

上から順に条件判定して初めて成立した部分の命令だけ実行

条件式①②が両方成立していた場合でも，命令文①だけ実行される

逆順はダメ

Shiftキー押しながら ￥を2回 → ||

条件式に使える比較演算子

演算子	条件式の例	成立条件
>	a > b	aがbより大
<	a < b	aがbより小
>=	a >=b	aがb以上
<=	a <= b	aがb以下
==	a == b	aとbが等しい
!=	a != b	aとbが異なる
&&	(条件式)&&(条件式)	両条件が成立
||	(条件式)||(条件式)	どちらかでも成立

条件式が成立している間繰り返したいときはwhile

書くのは簡単．
無限ループに入ると怖い．

whileを使った制御構造のテンプレート

条件分岐に使えるものと同じ

```
while(反復条件式){
    命令文;
}
```

ここで条件式に含まれる変数が変わる

反復条件が成立している間は実行される

9を超えないまで繰り返す例

aが9ならギリギリ反復実行

```
int a = 0;
while(a <= 9){
    printf(“%d ”, a);
    a = a+2;
}
```

➡ 「0 2 4 6 8」が表示される

決まった回数繰り返したいときはfor

よく使うのにややこしい

変数名は何でもよいがi,j,kがよく使われる

forを使った制御構造のテンプレート

< か <= を使う

基本的には反復ごとにカウンタを1ずつ増やす

```
for(カウンタ用変数の定義と初期化; 反復条件式; カウンタ動作){
    命令文;
}
```

反復条件式が満たされている間は繰り返し実行される

5回反復して命令を実行させる例

カウンタを1から始めて反復ごとに1ずつ増やし，5以下である間は繰り返す

```
int a = 0;
for(int i = 1; i<=5; i++){
    printf("%d ", a);
    a = a+2;
}
```

➡ 「0 2 4 6 8」が表示される