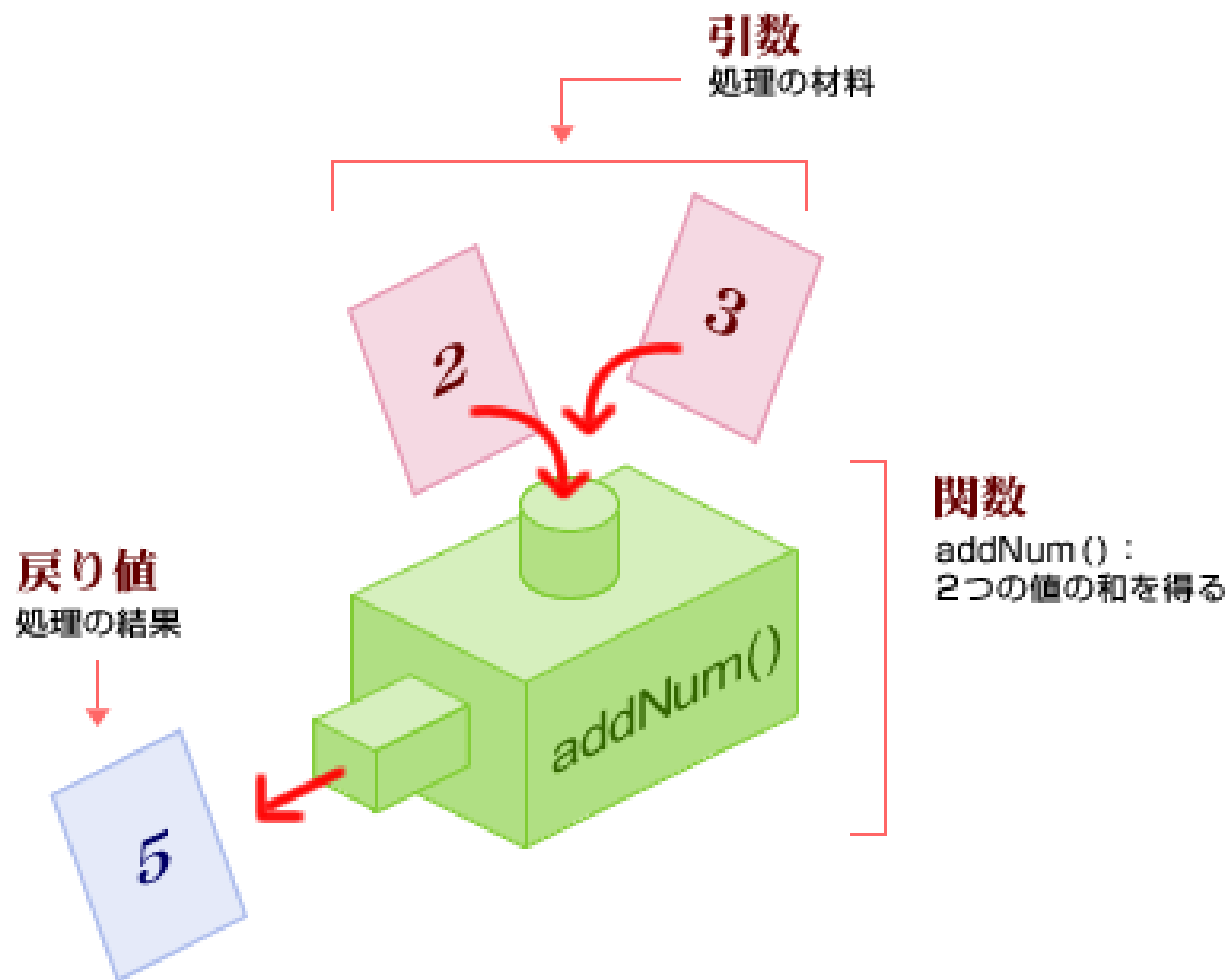


# プログラマー勉強会第3回



# 今回の作業ディレクトリ

Terminalを起動



cd Desktop/pandd\_prog/ex03 を入力

今回のスライドをpdfにしたものを

OneDriveの

\_2018/勉強会/プログラマー

のディレクトリに入れてあるので、  
ダウンロードして閲覧してください。

前回の内容を忘れてしまった人は、同じ場所に前回のスライドのpdfがあるのでそれを見てください。

# 何度登場する処理

## [Square.c]

```
#include <stdio.h>
int main(){
    int a = 2,b = 5,c = -3;
    int n;

    n = a * a;
    printf("a^2 = %d¥n",n);
    n = b * b;
    printf("b^2 = %d¥n",n);
    n = c * c;
    printf("c^2 = %d¥n",n);
    n = (a + b) * (a + b);
    printf("(a + b)^2 = %d¥n");
    n = (b + c) * (b + c);
    printf("(a + b)^2 = %d¥n");
    return 0;
}
```

左のプログラムは、  
a,b,c,a + b,b + cの2乗を計算し、その結果を表示するプログラムである。

が、変数が違うだけで同じような処理が何度も登場している。  
(nに2乗計算を格納 → 結果表示)

# プログラムの部品化

```
[Reuse.c]
```

```
#include <stdio.h>
```

```
void Square(int);
```

```
int main(){
```

```
    int a = 2,b = 5,c = -3;
```

```
    Square(a);
```

```
    Square(b);
```

```
    Square(c);
```

```
    Square(a + b);
```

```
    Square(b + c);
```

```
    return 0;
```

```
}
```

```
void Square(int s){
```

```
    int n;
```

```
    n = s * s;
```

```
    printf(“%d¥n”,n);
```

```
}
```

左のプログラムは、Square.cの

「nに2乗計算を格納 → 結果表示」  
の部分部品化したものである。

「nに2乗計算を格納 → 結果表示」  
の部分部品をSquareの中にまとめ、()の中に値を入れることで、処理を実行している。

このようにプログラムの処理を部品化したものを**関数**という。

# 関数

関数・・・まとまりのある処理を簡単に呼び出せるようにひとまとめにしたもの。  
今までツールと呼んでいた、`printf`、`scanf`も関数の一種。

## [関数の宣言]

関数も変数と同様に、使うために宣言が必要。

型名 関数名(引数の型名);

↑これをmain関数の上に記述する。

※例

***int function(int);***

→functionという名前のint型の関数。引数の型はint。

引数・・・関数に渡す数値。

$$f(x) = x^2 + 2x \quad \longrightarrow \quad f(3) = 15 \quad (x = 3)$$

↑これ

# 関数②

## [関数の定義]

宣言した関数がどのような処理をするのか。関数の中身を書く必要がある。

```
関数の型名 関数名(引数の型名 引数の変数名){  
    プログラム;  
    . . .  
}
```

## ※例

```
int function(int x){  
    int y;  
    y = x * x + 2 * x;  
    return y;  
}
```

→  $f(x) = x^2 + 2x$  をプログラムで書いたもの

# 関数③

## [関数内で変数の宣言]

main関数のように、各々自分で定義した関数内でも、先頭部分に変数を宣言できる。但し、これらの変数の有効範囲は宣言した関数の中だけ。main関数でもそれは同様。

↓これも一応変数宣言の一種

```
int function(int x){  
    int y; ← ここ  
    y = x * x + 2 * x;  
    return y;  
} ← 変数x,yの有効範囲はここまで
```

# 関数④

[変数は宣言された関数内でしか機能しない]

```
int main(){
```

```
    int x,y; ← 変数 x,y を宣言
```

```
    x = 3; ← xに3を代入 function内のxには変化なし
```

```
    y = function(3);
```

```
    return 0;
```

```
}
```

```
int function(int n){
```

```
    int x,y; ← main関数で宣言された変数と同名同型の変数を宣言
```

```
    y = 5; ← yに5を代入 main内のyには変化なし
```

```
    x = n * y;
```

```
    return x;
```

```
} ← 関数終了。ここで宣言された変数x,yの存在も消滅。main関数の変数には影響なし
```

このように同名同型の変数でも、宣言されている関数の場所が違えば全く別物となる。

また、関数をもう一度使用する時も、前に使用した時の変数の値は残っていない。



# 関数⑤

`return 値;` . . . 関数を終了させ、関数に値を持たせる。

```
int function(int x){  
    int y;  
    y = x * x + 2 * x;  
    return y;  
}
```

→引数を元に計算した結果`y`の値が、関数`function`の値になる。

→`x`が3の時、15が関数`function`の値になる。

値を持った関数は数値と同じ扱いになる。

変数に格納する

→ `int F;`  
`F = function(5);` ← `F`に35が代入される

値を表示する

→ `printf("function(4) = %d\n",function(4));`

# 関数⑥

## [関数に持たせる値の場合分け]

return で関数に値を持たせるとき、if文などを用いて持たせる値の場合分けをすることができる。

```
int unit_step(int n){  
    if(n >= 0)  
        return 1;  
    else  
        return 0;  
}
```

→ 引数の値が0以上の時は1、0未満の場合は0を関数に持たせる関数unit\_step(int);

```
double frac(double x){  
    double y;  
    if(x + 3 == 0)  
        return 0; ← x + 3(分母)が0の時、関数に0を持たせて終了。ここより先は実行されない。  
  
    y = x / (x + 3);  
    return y;  
}
```

→  $x / (x + 3)$ の値を出す関数。分母が0になる時は即終了する。

# 関数⑦

## [値を持たせないとき]

ただ関数内に記述された処理を与えられた引数を元に行うだけで、関数そのものに値を持たせない場合は、関数の型名に**void**を使う。returnは置かない。値を持っていないので、変数に格納することはできず、表示もできない。ただ関数内の処理を実行するだけ。

## ※例

```
void Square(int x){  
    int n;  
    n = x * x;  
    printf(“%d¥n”);  
}
```

# 関数⑧

## [関数の使用]

関数を宣言し、内容を定義したので関数を使用できる。  
使用したい場所に、関数名と引数に入れる値を記述する。

```
#include <stdio.h>
```

```
void Square(int); ← 関数の宣言
```

```
int main(){
```

```
    int a = 2;
```

```
    Square(a); ← 引数にa (2) を入れてSquare関数を使用している。
```

```
    return 0;
```

```
}
```

```
void Square(int s){ ← 関数の定義 引数の変数sには使用した時に入れた引数の値が代入される
```

```
    int n;
```

```
    n = s * s;
```

```
    printf(“%d¥n”,n);
```

```
}
```

# 演習

if～else文などを用いて、

$$f(x) = |x|$$

を満たす関数(`int absolute(int)`)を作成し、`main`関数内で様々な値を引数として使用して、5回使用してその値を表示するプログラムを作成してください。

但し、定義する関数内で`printf`は使用しないこと。

値の表示は`main`内で行う。(ファイル名 `ex3-1.c`)

[ヒント]

`x`が0以上の値の場合 → 引数の値をそのまま関数に持たせる。

`x`が0未満の場合 → 引数の値に-1を掛けて関数に持たせる。

# 演習の答え

```
#include <stdio.h>
```

```
int absolute(int);
```

```
int main(){
```

```
    printf("absolute(3) = %d\n",absolute(3));
```

```
    printf("absolute(-3) = %d\n",absolute(-3));
```

```
    printf("absolute(0) = %d\n",absolute(0));
```

```
    printf("absolute(810) = %d\n",absolute(810));
```

```
    printf("absolute(-931) = %d\n",absolute(-931));
```

```
    return 0;
```

```
}
```

```
int absolute(int x){
```

```
    if(x >= 0)
```

```
        return x;
```

```
    else
```

```
        return -x;
```

```
}
```

# 関数⑨

## [引数を複数持たせる]

数学の  $f(x, y, z)$  ように、関数に引数を複数持たせることもできる。  
引数を, で区切って記述する。

## ※例

```
double sum(double x,double y,double z){  
    double S;  
    S = x * x + y * y + z * z;  
    return S;  
}
```

→3つの数値の2乗の合計を求める。

呼び出し時

→ `sum(3.2,6.4,-2.5);`

# 演習

台形の面積を求める関数(*`double trapzoid(double, double, double)`*)を定義して、`main`関数内で使用して、面積の値を表示するプログラムを作ってください。  
引数は右から順に、高さ、上底、下底とすること。  
高さ、上底、下底の値は `scanf` を使って入力から代入すること。  
(ファイル名 `ex3-2.c`)

$$\text{台形の面積} = (\text{上底} + \text{下底}) \times \text{高さ} \div 2$$



# 演習の答え

```
#include <stdio.h>
```

```
double trapezoid(double, double, double);
```

```
int main(){  
    double u, d, h;  
    printf("値を入れてください：");  
    scanf("%lf%lf%lf", &h, &u, &d);  
    printf("台形の面積は%d¥n", trapezoid(h, u, d));  
    return 0;  
}
```

```
double trapezoid(double h, double u, double d){  
    double S;  
    S = (u + d) * h / 2;  
    return S;  
}
```

# 関数⑩

## [引数のない関数]

数学ではあり得ないが、引数のない関数もプログラミングでは存在する。  
値を取り込まず、一定の処理をする時などに使う。

()の中に何も記述しない、もしくはvoidと書く。

※例

```
void Rotate(){  
    int i;  
  
    for(i = 0;i < 16;i++){  
        printf("クソ");  
        printf("ギョルルルルルル¥n");  
    }  
}
```



→右の画像のセリフを出力する関数

呼び出し時

→ Rotate();

# 演習

for文、if～else文、printfなどを使って、文字列を5行表示し、  
奇数行には 「☆ ☆ ☆」 を  
偶数行には 「 ☆ ☆ 」 を表示する関数(void star())を定義し、  
main内で使用するプログラムを作ってください。(ex3-3.c)

うまくいくと以下のように表示される。

```
☆ ☆ ☆  
  ☆ ☆  
☆ ☆ ☆  
  ☆ ☆  
☆ ☆ ☆
```

ヒント

for文のカウンタ変数は1から始め、5になるまでループさせる。

偶数行 → カウンタ変数が偶数 → カウンタ変数%2 が0になる時

奇数行 → カウンタ変数が奇数 → カウンタ変数%2 が1になる時

# 演習の答え

```
#include <stdio.h>
```

```
void star();
```


```
int main(){  
    star();  
    return 0;  
}
```

```
void star(){  
    int i;  
    for(i = 1;i <= 5;i++){  
        if(i % 2 == 1)  
            printf("☆  ☆  ☆¥n");  
        else  
            printf("  ☆  ☆  ¥n");  
    }  
}
```

# ファイル进行操作する

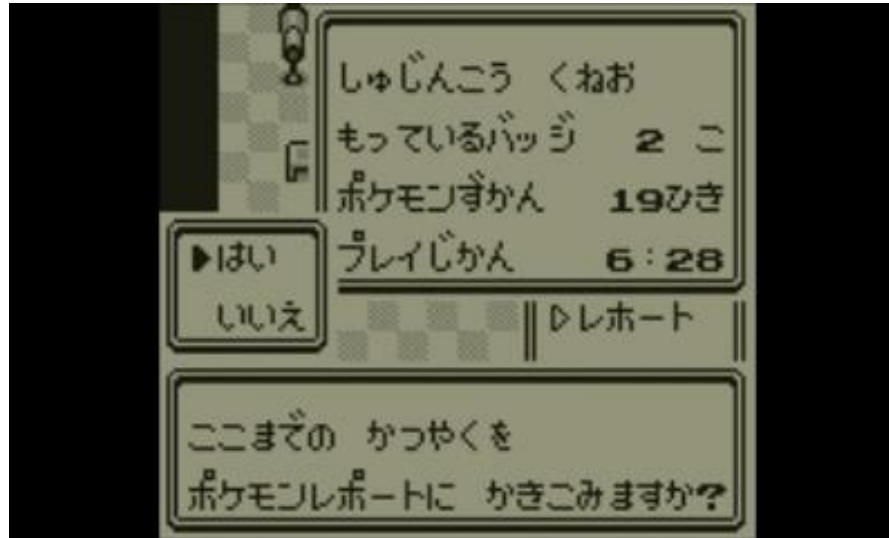
今までのプログラムは、プログラム内にあらかじめ数値を記述するか、scanfを使って値を入力して動作させてきた。

が、プログラム外から数値を読み込んだり、プログラム内で生成した数値をプログラム終了後も保存するために、ファイルに書き込んだりする動作も必要になってくる。

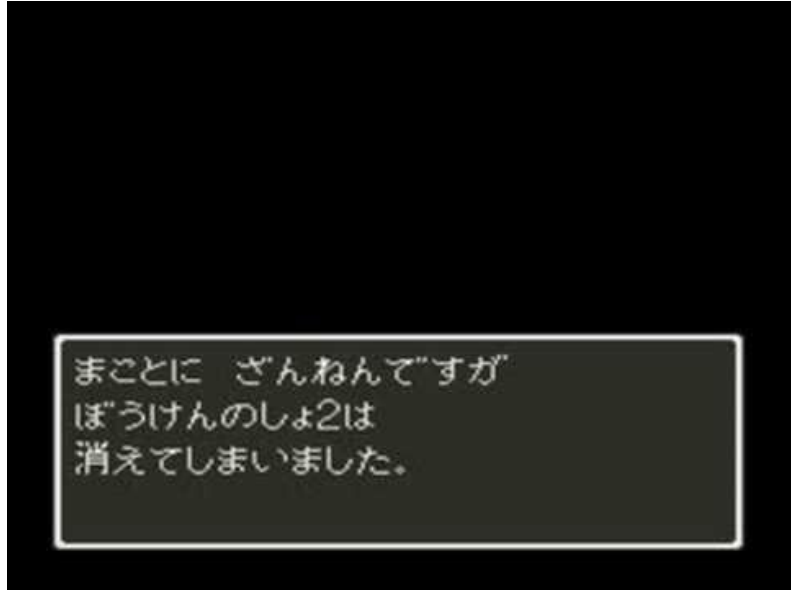


NOW LOADING

データ読み込み



データ書き込み



まごとに さんねんですが  
ぼうけんのしょ2は  
消えてしまいました。

ファイルに書き込めないとうなる

# ファイルから情報を読み込む

**sample.txt** というファイルを作成し、その中に好きな整数を3つスペースで区切って書く。

**[read.c]**

```
#include <stdio.h>

int main(){
    FILE *read;
    int a,b,c;
    read = fopen("sample.txt","r");
    fscanf(read,"%d%d%d",&a,&b,&c);
    printf("%d %d %d",a,b,c);
    fclose(read);
    return 0;
}
```

実行すると、sample.txtに書かれている数値が表示される。

# ファイル読み込みの流れ

FILE \*read . . . ファイルポインタ変数。ファイルの場所を記憶する特殊な変数。

fopen("sample.txt","r") . . . ファイルを開く関数。ファイルポインタ変数に格納できる値を持つ。  
左の引数で開くファイルを指定し、  
右の引数でファイルに対する操作を指定する。

**ファイルポインタ変数 = fopen("ファイルへのパス","ファイルに対する操作");**

ファイルに対する操作 . . . rが読み込み、wが書き込みを表す。  
読み込みの際、該当するファイルがない場合、ファイルポインタ変数に「NULL」という値が入る。  
NULLの状態、ファイルを参照しようとするエラーが起きる。

fscanf(read,"%d%d%d",&a,&b,&c); . . . ファイル読み込み用scanf。  
ファイルに書かれている値を変数へ代入していく。  
最初の引数にファイルポインタを入れる以外はscanfと同じ。

fclose(read) . . . fopenで開いたファイルを閉じる関数。

# ファイルへ情報を書き込む

**[write.c]**

```
#include <stdio.h>
```

```
int main(){  
    FILE *write;  
    int a = 3,b = 2,c = -1;  
    write = fopen("write.txt","w");  
    fprintf(write,"%d %d %d\n",a,b,c);  
    fclose(write);  
    return 0;  
}
```

実行すると、`write.txt`というファイルが作成され、変数`a,b,c`の値が書き込まれる。



# ファイル書き込みの流れ

`fopen("write.txt","w");` . . . 書き込みなので、ファイルに対する操作の引数をwにする。

書き込みの際、該当するファイルがない場合、新たにファイルを作成して開く。  
すでにある場合はファイルの中身を全て削除し、ファイルを開く。

`fprintf(read,"%d%d%d",&a,&b,&c);` . . . ファイル読み込み用printf。  
2番目の引数の文字列をファイルへ書き込む。

**fopen,fscanf,fprintf,fcloseは全てstdio.hの中に入っている関数である。**

# 読み込んで書き込む

**sample.txt**に適当な整数をスペースで区切って3つ書き込む

## [ReadAndWrite.c]

```
#include <stdio.h>
```

```
int main(){
    FILE *read,*write;
    int data[3],i;
    read = fopen("sample.txt","r");
    write = fopen("result.txt","w");

    for(i = 0;i < 3;i++)
        fscanf(read,"%d",&data[i]);
    for(i = 0;i < 3;i++)
        data[i] += 5;
    for(i = 0;i < 3;i++)
        fprintf(write,"%d",data[i]);
    fclose(read);
    fclose(write);
    return 0;
}
```

実行すると、**sample.txt**に書かれている値に5を足した値が**result.txt**に書き込まれる。

# 演習

input.txtというファイルを作成し、そこに適当な整数**10**個をスペースで区切って書き込む。  
これを読み込んで、**10**個の整数の合計の値をoutput.txtというファイルに書き込むプログラムを作ってください。(ファイル名：**ex3-4.c**)

# 演習の答え

```
#include <stdio.h>
```

```
int main(){
```

```
    FILE *read,*write;
```

```
    int i,data[10],S = 0;
```

```
    read = fopen("input.txt","r");
```

```
    write = fopen("output.txt","w");
```

```
    for(i = 0;i < 10;i++)
```

```
        fscanf(read,"%d",&data[i]);
```

```
    for(i = 0;i < 10;i++)
```

```
        S += data[i];
```

```
    fprintf(write,"%d",S);
```

```
    fclose(read);
```

```
    fclose(write);
```

```
    return 0;
```

```
}
```

# 複数の変数をひとまとめにする

ゲーム制作では多くの変数や配列が必要になってくる。  
プレイヤーだけでも5個以上の変数が必要となる。  
それらを全て宣言しても良いが、かさばって見づらいプログラムになりやすい。



[プレイヤーに必要な情報]

X座標

Y座標

高さ

幅

ライフ

角度

etc...

これらの情報を一つにまとめて取り扱えるようにするものとして、  
**構造体**というものがある。

# 構造体と新たな型作り

**構造体**・・・複数の変数や配列をひとまとめにして、新たな変数の型として使うもの

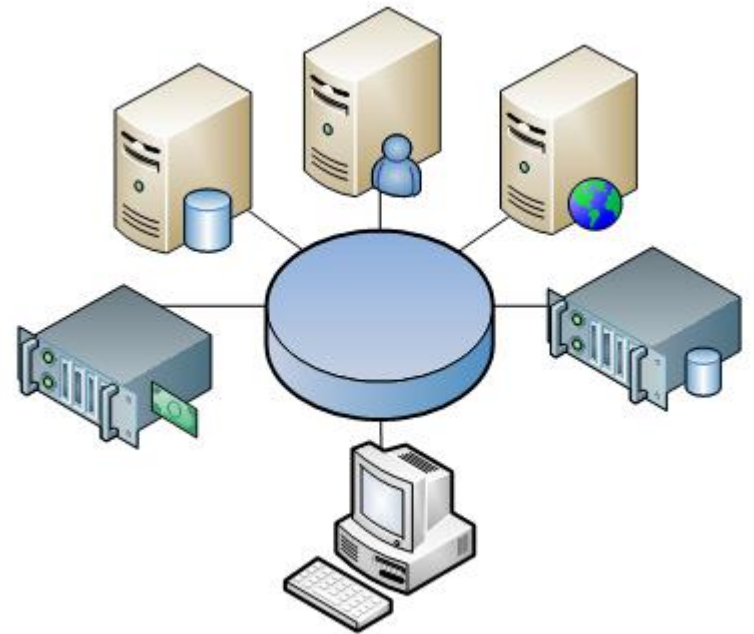
## [定義のやり方]

```
typedef struct{  
    構造体にまとめておく変数や配列;  
    . . .  
}構造体の名前;  
→これをmain関数の上に記述する
```

## [定義例]

プレイヤーの情報をまとめる構造体PLAYERを定義

```
typedef struct{  
    int X;  
    int Y;  
    int W;  
    int H;  
    int life[10];  
    double angle;  
}PLAYER;
```



# 構造体を変数として宣言

定義した構造体は、intやdoubleのように変数として宣言したり値を保存したりできる。

## [構造体の宣言]

構造体名 構造体変数名;

## [宣言例]

PLAYER pl;

→通常の変数と全く同じ

→配列の場合も同様 → PLAYER pl\_array[10];

## [構造体の初期化]

構造体の中に入っている変数へ、宣言時に値を代入していく

構造体名 構造体変数名 = {値,値,値,...値};

→上から順に入っていく

## [初期化例]

PLAYER pl = {20,40,100,40,{5},20.8};

→構造体PLAYERの中の、Xに20が、Yに40が、Wに100が、Hに40が、配列lifeの全要素に5が、angleに20.8が入る。

# 構造体の中の変数を参照

構造体変数そのままでは、中の変数の値を参照することはできない。  
構造体の中の変数を取り出すために、**メンバ演算子**というものを使う。

## [メンバ演算子の使い方]

構造体変数名.構造体の中の変数;

→構造体変数と参照したい構造体の中の変数を「.(ドット)」でつなぐ。

## [参照例]

```
PLAYER pl;
```

```
pl.X = 20; ← PLAYER構造体の中のXを参照してXに20を代入する。
```

```
pl.life[3]--; ← 構造体の中の配列lifeの3番目をデクリメントする。
```

```
scanf("%lf",&pl.angle); ← 構造体の中のangleの値をscanfを使って代入。
```

```
printf("%d%d¥n",pl.W,pl.H); ← 構造体の中のW、Hの値を表示。
```



# 構造体同士の代入

同じ型の構造体同士であれば、代入演算子 = を使うことができる。

## [例]

```
PLAYER pl1,pl2;
```

```
pl1 = pl2;
```

↑pl2の中の変数の値がすべてpl1の中の変数へと代入される。

同じ型であれば代入が行えるため、関数の引数や型に使用することもできる。

```
→ void Player_Struct(PLAYER);
```

```
→ PLAYER Ret_Player(int);
```

# 演習

exam.txt というファイルを作り、以下の内容を入力。

89 57 77

100 0 23

56 78 22

77 88 12

76 89 49

このファイルは5人の生徒のテストの点数を表したものである。

このファイルを読み込んで、各生徒の3科目の平均点を求め、表示するプログラムを作ってください。

## [条件]

- ・ 生徒の情報(各テストの点数、平均点)を示す変数は**Student**という構造体を作って、そこに格納すること。
- ・ 平均点を求める処理は関数化すること。 (*void Eval(Student)*)
- ・ 表示も関数内で行うこと。

# 演習の答え

```
#include <stdio.h>
```

```
typedef struct{  
    int Sub1;  
    int Sub2;  
    int Sub3;  
    double Eval;  
}
```

```
void Eval(Student);
```

```
int main(){  
    Student data[5];  
    FILE *read;  
    int i;  
  
    read = fopen("Exam.txt","r");  
    for(i = 0;i < 5;i++)  
        fscanf(read,"%d%d%d",&data[i].Sub1,&data[i].Sub2,&data[i].Sub3);
```

# 演習の答え

```
for(i = 0;i < 5;i++)  
    Eval(data[i]);
```

```
return 0;
```

```
}
```

```
void Eval(Student data){  
    data.Eval = (data.Sub1 + data.Sub2 + data.Sub3) / 3;  
    printf(“%f¥n”,data.Eval);  
}
```

おわり