

# 情報研究会CACTUS

第4回 講義資料

# 今回の内容

- ・ 前回のおさらい
- ・ 関数
- ・ 変数のスコープ
- ・ 再帰関数

# おさらい

配列とは？

変数を格納する箱を列に並べたもの。

ひとつひとつに変数を格納できる。収納上手！

例) `int a[10];`

配列	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
変数	693	12	-1444	114	514	1919	-25	564	5	1562

# おさらい

関数とは？

main関数の中身の一部をパーツにわけたもの。

自分で新しい関数（機能）がつかれる！

基本、プロトタイプ宣言をする必要がある。

# おさらい問題

次のコードの間違いを探せ！（3か所）

```
#include<stdio.h>
void printout(void)

int main(void)
{
    int array[10];
    array[10] = 114514;
    scanf("%d",array[5]);

    printout();
    return 0;
}

void printout() {
    printf("私の名前はねこ¥n");
    printf("とってもやさしいご主人様のねこなの¥n");
}
```

# サンプルコード1

実引数（上）、  
仮引数（下）の  
変数名が違うが  
きちんと動く。

```
1  #include<stdio.h>
2  int summation(int, int, int);
3
4  int main(void) {
5      int a, b, c, sum;
6
7      printf("整数を3つ入力せよ\n");
8      scanf("%d%d%d", &a, &b, &c);
9
10     sum = summation(a, b, c);
11
12     printf("合計は%dです\n", sum);
13     return 0;
14 }
15
16 int summation(int x, int y, int z) {
17     int goukei;
18     goukei = x + y + z;
19     return goukei;
20 }
```

# 関数の宣言

戻り値の型名 関数名 (型 引数1, 型 引数2...) {}

と宣言する。具体的には、

```
void kansuu(int a, int b){  
    printf(" this is kansuu" );  
}
```

→ 引数とは？

→ 戻り値とは？

voidとは  
「空、なにもない」という  
意味。  
つまり、戻り値がない  
ということ。

# 引数とは

作成した関数とmain関数の中にある変数の値を共有するために必要なもの。

仮引数→関数、プロトタイプ宣言

実引数→main内

仮引数は関数内部で使われ、main内の変数（実引数）の値を格納する。

ただし、逆は起きない（詳しくは後述）。



# 実引数とは

関数に渡す値のこと。

関数はいくまでもパーツなので、何かを計算させるために具体的な値をあげる必要がある。

関数というならば「鑄型」、その中に注ぎ込む素材が実引数。



# 仮引数とは

関数内部で使われる変数を格納するもの。

プロトタイプ宣言では、型だけを宣言してもよい。

この値は関数内で使われた後に実引数に渡されることはないので注意。

また、仮引数と実引数の変数名は同一でなくともよい。

# 戻り値とは？

関数内で行った処理の結果をmain関数に戻したいときに使われる値。関数を行った結果。

チョコレートの例でいうと、完成したチョコレートをmain関数にあげる。つまりバレンタインのことである。

例)

```
int catcount(int kuro,int siro){  
    int goukei;  
    goukei = kuro + siro;  
    return goukei;  
}
```

この値をmainに返す。  
goukeiはint型なので、宣言も  
int型になっている

# 関数の呼び出し方

main関数で作成した関数を使うことを、「呼び出す」という。

戻り値がある場合とない場合で違いがある。

ある場合    `D = cat(a,b);`  
              `printf(" %d¥n" ,cat(a,b));`

ない場合    `catcatcat(a,b,c);`

printfで表示したいときはそのまま関数を入れても可能。

(a+bをそのままprintfに入れるのと同じようなこと)

# 処理の流れ

x=a;y=b;z=c;

goukei=x+y+z;

sum= goukei;

```
1  #include<stdio.h>
2  int summation(int, int, int);
3
4  int main(void) {
5      int a, b, c, sum;
6
7      printf("整数を3つ入力せよ\n");
8      scanf("%d%d%d", &a, &b, &c);
9
10     sum = summation(a, b, c);
11
12     printf("合計は%dです\n", sum);
13     return 0;
14 }
15
16 int summation(int x, int y, int z) {
17     int goukei;
18     goukei = x + y + z;
19     return goukei;
20 }
```

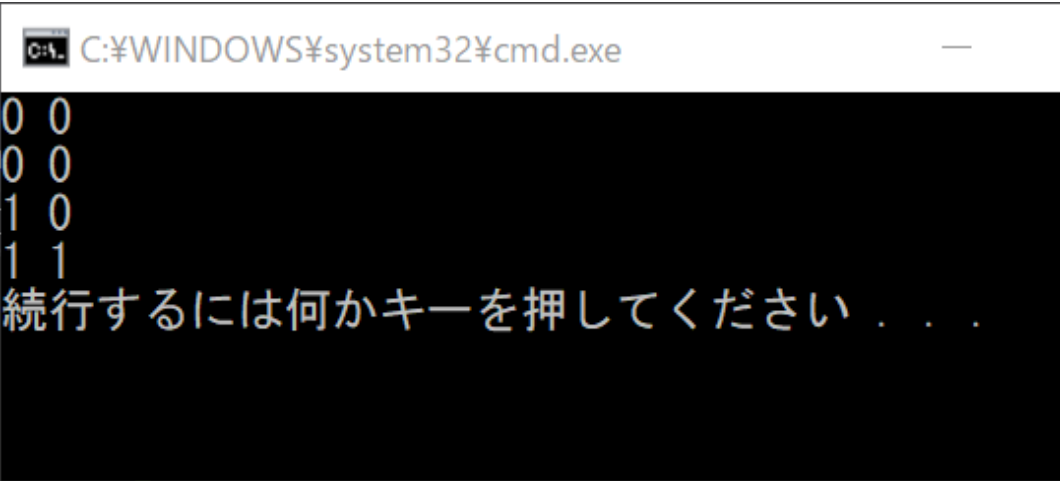
# 確認問題1

サンプルプログラムに $a*b*c$ の値を返す関数を新しく作成して追加してください。

# サンプルコード2

```
1  #include<stdio.h>
2  void printXY(int x, int y);
3
4  x = 0;
5
6  int main(void) {
7      int y = 0;
8
9      printXY(x, y);
10     printXY(x, y);
11     x++;
12     printXY(x, y);
13     y++;
14     printXY(x, y);
15
16     return 0;
17 }
18
19 void printXY(int x, int y) {
20     printf("%d %d\n", x, y);
21     x++;
22     y++;
23 }
```

## 実行結果



```
C:\WINDOWS\system32\cmd.exe
0 0
0 0
1 0
1 1
続行するには何かキーを押してください . . .
```

# 変数のスコープ

変数には**通用する範囲**があり、スコープと呼ぶ。

## グローバル変数

プログラム内すべてがスコープである変数。

{ }で囲われていない場所で宣言できる。

## ローカル変数

{ }で囲われている内部がスコープである変数。

つまりブロック内部がスコープである。



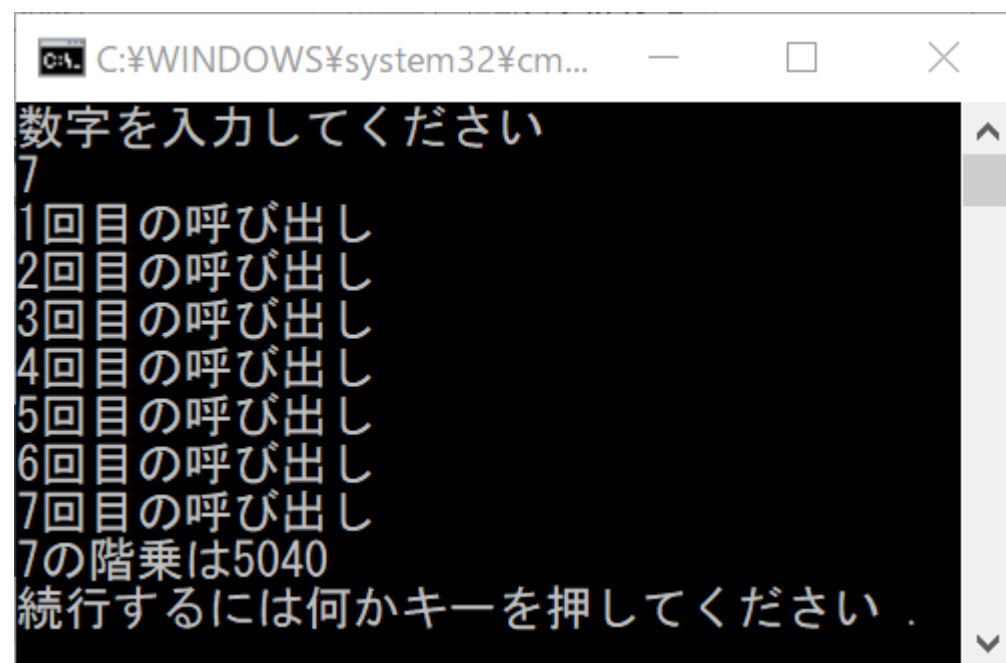
# ちがひ

**グローバル変数**はどこでも値を変えられるため、間違えた値を代入したときに間違いを発見しづらい。

**ローカル変数**は{}で囲われた部分でしか通用しないため、他の関数でも同じ名前を使用することができる。そのためどちらの関数の変数なのかが混同しやすい。また、ローカル変数は{}内の処理がすべて終わったときにメモリ上から消去される。

# サンプルコード3

nの階乗を  
求める  
プログラム



```
C:\WINDOWS\system32\cmd.exe
数字を入力してください
7
1回目の呼び出し
2回目の呼び出し
3回目の呼び出し
4回目の呼び出し
5回目の呼び出し
6回目の呼び出し
7回目の呼び出し
7の階乗は5040
続行するには何かキーを押してください .
```

```
1  #include<stdio.h>
2  int kaijo(int);
3
4  int main() {
5      int n;
6
7      printf("数字を入力してください\n");
8      scanf("%d", &n);
9
10     printf("%dの階乗は%d\n", n, kaijo(n));
11     return 0;
12 }
13
14 int kaijo(int n) {
15     static int count = 1; //静的ローカル変数
16     printf("%d回目の呼び出し\n", count);
17     count++;
18
19     if (n == 1) return 1;
20
21     return n*kaijo(n - 1);
22 }
```

# 静的ローカル変数

ローカル変数とは違い、メモリ上から解放されない。  
つまり、関数を何度呼び出しても値が初期化されずに保存されたままとなる。

宣言方法は初めに「static」をつける。

例)

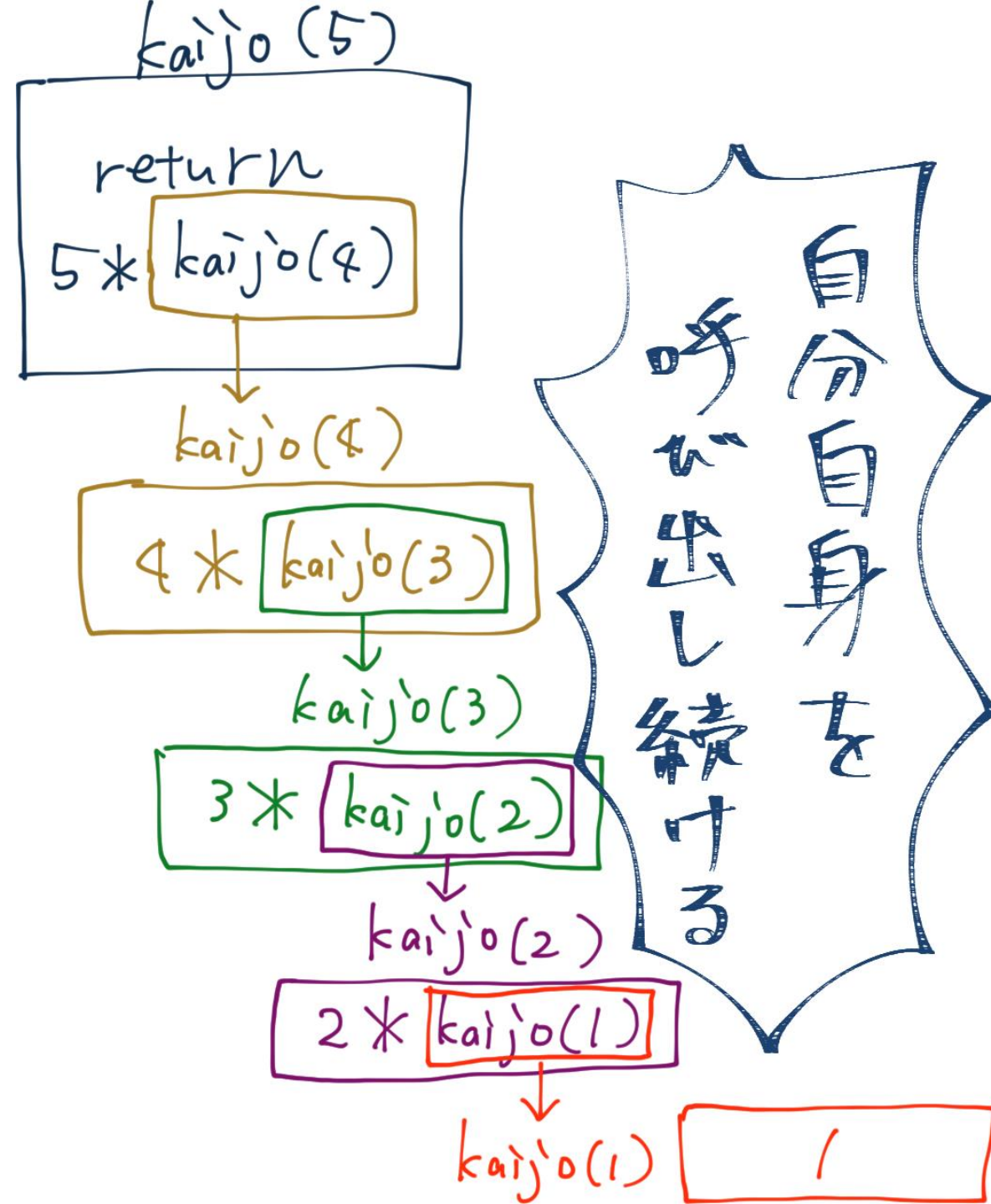
```
static int neko;
```

サンプルコードのstaticを外して実行してみよう！

# 再帰関数

関数の内部で、  
自分自身の関数を  
呼び出すこと。

これを再帰呼び出しと  
呼び、それをする関数を  
再帰関数と呼ぶ。



# 確認問題1

再帰関数を用いて、1から入力された整数までの総和を求めるプログラムを作成してください。  
(サンプルコードを参考にするとよい)

# 練習問題1

5つの入力された数を半径とする円の円周及び面積をもとめるプログラム。

17行目の

ここにコードを書く

と書かれた部分に  
半径と円周を求める  
プログラムを作成せよ。

```
1  #include<stdio.h>
2  void keisan(double);
3  #define PI 3.14 //円周率
4
5  int main() {
6      double a[5]; //入力された数字を入れておく配列
7      int i; //カウンタ変数
8
9      printf("数字を入力してください\n");
10
11     for (i = 0; i < 5; i++) {
12         scanf("%lf", &a[i]); // 配列に半径を記録させる
13     }
14     for (i = 0; i < 5; i++) {
15         keisan(a[i]); //半径と面積を求めて表示する
16     }
17
18     return 0;
19 }
20
21 //半径と面積を計算して表示するプログラム
22 void keisan(double n) {
23     //ここにコードを書く
24 }
```

# 練習問題2

フィボナッチ数列の第 $n$ 項を再帰関数を用いて表示するプログラム。

フィボナッチ数列の漸化式

$$f(n+1) = f(n) + f(n-1)$$

$$f(1) = 1, f(2) = 1$$

1, 1, 2, 3, 5, 8, 13, 21...となる。

# 練習問題3

学籍番号のチェックディジットを求めるプログラム

(参考) チェックディジットの求め方

1. 学籍番号の各桁に2から7を掛けて足し合わせる
2. 足し合わせた数を11で割った余りに対応する  
アルファベットを変換表から探す

例) 152910の場合

$$1 \times 7 + 5 \times 6 + 2 \times 5 + 9 \times 4 + 1 \times 3 + 0 \times 2 = 86 \quad 86 \% 11 = 9 \quad 9 \rightarrow C$$

変換表

0	→	A
1	→	A
2	→	Z
3	→	Y
4	→	X
5	→	U
6	→	M
7	→	K
8	→	H
9	→	C
10	→	B

学籍番号  
152910C