# 情報研究会CACTAS

## 第八回　講義資料

# 練習問題１の解答

```
1  #include<stdio.h>
2
3  int main(void){
4          int arr[10] = {1,2,3,4,5,6,7,8,9,0} , arr2[10];
5          int *p1 = arr,*p2=arr2;
6          int i;
7
8          p1+=9;
9          for(i=0;i<=9;i++){
10                 *p2 = *p1;
11                 printf("%d",*p2);
12                 p1--;
13                 p2++;
14
15         }
16
17         printf("¥n");
18
19 }[EOF]
```

# 練習問題２の解答

```
1  #include<stdio.h>
2  #define N 20
3
4  int main(void){
5          int arr[N];
6          int *p=arr;
7          int i,n;
8
9          printf("数字を入力してください¥n");
10         for(i=0;i<=9;i++){
11                 scanf("%d",&arr[i]);
12         }
13         while(1){
14                 printf("何番目の数を出力しますか：¥n");
15                 scanf("%d",&n);
16                 p+=n-1;
17                 printf("%d番目の数は%d¥n",n,*p);
18                 p-=n-1;
19         }
20         return 0;
21 }[EOF]
```

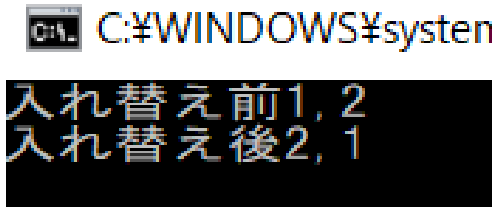
# 練習問題３の解答

```c
#include<stdio.h>
#define N 100

int main(void){
        int arr[N] = {1,1};
        int *p;
        int n,i;
        p=arr;

        printf("何番目の項を表示しますか：");
        scanf("%d",&n);

        if(n==1||n==2){
                printf("1¥n");
        }else if(n>=3){
                p+=2;
                for(i=3;i<=n;i++){
                        *p=*(p-1) + *(p-2);
                        //printf("%d¥n",*p);
                        p++;

                }
                printf("%d¥n",*(p-1));
        }else{
                printf("不正な入力¥n");
        }
        return 0;
}[EOF]
```

# サンプルプログラム１

```c
#include<stdio.h>

void swap(int *p,int *q);

int main(void){
        int a=1,b=2;
        int *p,*q;
        p=&a;
        q=&b;

        printf("入れ替え前%d,%d¥n",a,b);
        swap(&a,&b);
        printf("入れ替え後%d,%d¥n",a,b);

        return 0;
}

void swap(int *p,int *q){
        int tmp_a,tmp_b;
        tmp_a = *p;
        tmp_b = *q;

        *p = tmp_b;
        *q = tmp_a;
}[EOF]
```

実行結果



# ポインタの関数引き渡し

ポインタを引数として関数に渡すとき、

- プロトタイプ宣言

データ型　関数名 (データ型 *ポインタ変数名);

- 変数使用時

関数名 (&ポインタ変数名);

関数の引数がポインタ変数であることを、* を付けてあらかじめ言っておく。

引数にはアドレスを入れること

```
#include<stdio.h>

void swap(int *p,int *q);

int main(void){
        int a=1,b=2;
        int *p,*q;
        p=&a;
        q=&b;

        printf("入れ替え前%d,%d¥n",a,b);
        swap(&a,&b);
        printf("入れ替え後%d,%d¥n",a,b);

        return 0;
}

void swap(int *p,int *q){
        int tmp_a,tmp_b;
        tmp_a = *p;
        tmp_b = *q;

        *p = tmp_b;
        *q = tmp_a;
}[EOF]
```

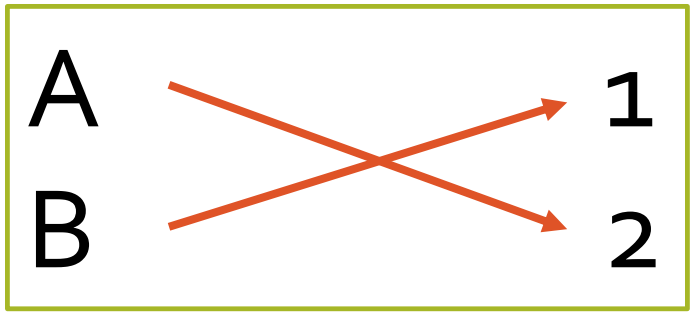
# サンプル解説

- Tmp_a = *p;　（又は、tmp_b=*q;）

  tmp_aにポインタpが指す変数（＝１）を代入する。

- *p = tmp_b　（又は、*q= tmp_a;）
- ポインタpが指す変数にtmp_b（＝２）を代入する。

- つまり、aとbが入れ替わったということ。




```c
#include<stdio.h>

void swap(int *p,int *q);

int main(void){
        int a=1,b=2;
        int *p,*q;
        p=&a;
        q=&b;

        printf("入れ替え前%d,%d¥n",a,b);
        swap(&a,&b);
        printf("入れ替え後%d,%d¥n",a,b);

        return 0;
}

void swap(int *p,int *q){
        int tmp_a,tmp_b;
        tmp_a = *p;
        tmp_b = *q;

        *p = tmp_b;
        *q = tmp_a;
}[EOF]
```

# 関数を使うメリット

- いままでは関数を使うことで戻り値は一つだけであった。
  しかし、ポインタを関数で使用することで、
  複数の変数を関数内で書き変えることができる！

- 例）　a=1　　→　　a=2
  　　b=2　　→　　b=1
  　　という二つの変数のデータの上書きを、
  　　一つの関数内で行うことができる。

```
17
18 void swap(int *p ,int *q){
19         int tmp_a,tmp_b;
20         tmp_a = *p;
21         tmp_b = *q;
22
23         *p = tmp_b;
24         *q = tmp_a;
25 }[EOF]
```

# サンプル２（書く必要はないです）

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

typedef struct{
        char name[100];
        int gakunen;     //学年
        int hit;         //ヒット
        int HR;          //ホームラン
        int tourui;      //盗塁
        double daritsu; //打率
        double taiju;    //体重
        double shincho; //身長
}baseball;

void pointer_print(baseball *data);//ポインタを引数にするときは*を付ける

int main(void){
        //構造体変数の宣言
        baseball data;
        //各データへのアクセス
        strcpy(data.name,"矢帆論子");
        data.gakunen    =2;
        data.hit        =2034;
        data.tourui     =32;
        data.HR         =54;
        data.daritsu    =0.410;
        data.taiju      =82.5;
        data.shincho    =186.0;

        //関数を呼びます
        pointer_print(&data);

        return 0;
}
```

```c
void pointer_print(baseball *data){
        printf("[氏名]%s¥n",(*data).name);
        printf("[学年]%d¥n",(*data).gakunen);//(*__)/___でアクセス
        printf("[ヒット]%d¥n",(*data).hit);
        printf("[盗塁]%d¥n",(*data).tourui);
        printf("[ホームラン]%d¥n",data->HR);//___->____でアクセス
        printf("[打率]%f¥n",data->daritsu);
        printf("[体重]%f¥n",data->taiju);
        printf("[身長]%f¥n",data->shincho);
}[EOF]
```

## 実行結果

```
[氏名]矢帆論子
[学年]2
[ヒット]2034
[盗塁]32
[ホームラン]54
[打率]0.410000
[体重]82.500000
[身長]186.000000
```

# サンプル２解説

- 構造体もポインタを使って処理することができる
- ポインタを使うメリットとは？

処理の高速化

構造体を関数に渡すと、全てのデータがコピーされる。
しかし、ポインタはアドレスを渡すだけなので、高速な処理が可能

# サンプル２解説

- 関数には構造体のポインタが
  渡されているので、

Data._____;

のようにはできないので、

(*Data)._____;

のように構造体にアクセスする。

ただし、

Data ->_____;

のうようなアクセスもできるし、手軽である。

```
36 void pointer_print(baseball *data){
37         printf("[氏名]%s¥n",(*data).name);
38         printf("[学年]%d¥n",(*data).gakunen);//(*__)/___でアクセス
39         printf("[ヒット]%d¥n",(*data).hit);
40         printf("[盗塁]%d¥n",(*data).tourui);
41         printf("[ホームラン]%d¥n",data->HR);//___->____でアクセス
42         printf("[打率]%f¥n",data->daritsu);
43         printf("[体重]%f¥n",data->taiju);
44         printf("[身長]%f¥n",data->shincho);
45 }[EOF]
```

# サンプル３

```c
#include<stdio.h>
int sum(int *data);

int main(void){
        int arr[5] = {54,53,84,45,26};
        int *data;
        //配列の先頭アドレスをデータに格納
        data =arr;
        //sum関数の戻り値を出力する
        printf("合計は、%d¥n",sum(data));
        return 0;
}

int sum(int *data){
        int i,sum=0;
        for(i=0;i<5;i++){
                //ポインタを配列の様に扱える！
                sum += data[i];
        }
        return sum;
}[EOF]
```

実行結果

# 配列とポインタ

- ポインタは配列の様に[]で要素を
  指定するような書き方ができる。

```
13
14 int sum(int *data){
15         int i,sum=0;
16         for(i=0;i<5;i++){
17                 //ポインタを配列の様に扱える！
18                 sum += data[i];
19         }
20         return sum;
21 }[EOF]
```

[]の役割について

[]は配列の要素番号を指定する演算子だが、

中身は配列のアドレスに足し算をしているだけである。

よって、一見別モノに見えるポインタにも[]を使って配列の様に表現できる。

# 練習問題

- サンプル１を利用して、二つの整数の和、差、積、商、余を出力するプログラムを組んでみよう。
- Void calc(int,int);という関数を自分で作ろう。
- このcalc関数内で和、差、積、商、余を計算し、メイン関数でそれぞれを出力しよう。
- 作れる関数は一つだけとする。

# 練習問題

- サンプル３を利用して、フィボナッチ数列の第ｎ項を求めるプログラムを組んでみよう。
- void fib(int *p)関数を自分で作ってみよう。
- 1 1 2 3 5 8 13 21 34 55 ….