関数と配列１再掲（教科書Step21 p.87 sample21_1.c 一部変更）

```
#include <stdio.h>
void sum_ave(int DATA[],int size,int sum[],float ave[])
{
        int i;
        sum[0]=0;
        for(i=0;i<size;i++)
        {
        sum[0]+=DATA[i];
        }/*位置変更*/
        ave[0]=(float)sum[0]/(float)size; /*位置変更*/
}
main()
{
        int data[5]={1,2,3,4,5},ans_sum[1];
        float ans_ave[1];
        sum_ave(data,5,ans_sum,ans_ave);
printf(”sum=%d ave=%f¥n”,ans_sum[0],ans_ave[0]);
}
```

実行例

```
$ ./a.out
sum=15 ave=3.000000
```

配列の先頭成分のアドレスのみ
を関数に引き渡している

ポインタと関数１（教科書Step24 p.99 sample24_1.c 一部修正）

```
#include <stdio.h>
void sum_ave(int DATA[],int size,int *sum,float *ave)
{
      int i;
      *sum=0;
      for(i=0;i<size;i++)
        {
         *sum+=DATA[i];
         }
      *ave=(float)*sum/(float)size;
}
main()
{
        int data[5]={1,2,3,4,5},ans_sum;
        float ans_ave;
        sum_ave(data,5,&ans_sum,&ans_ave);
printf(”sum=%d ave=%f¥n”,ans_sum,ans_ave);
}
```

配列の先頭成分アドレス引き渡し

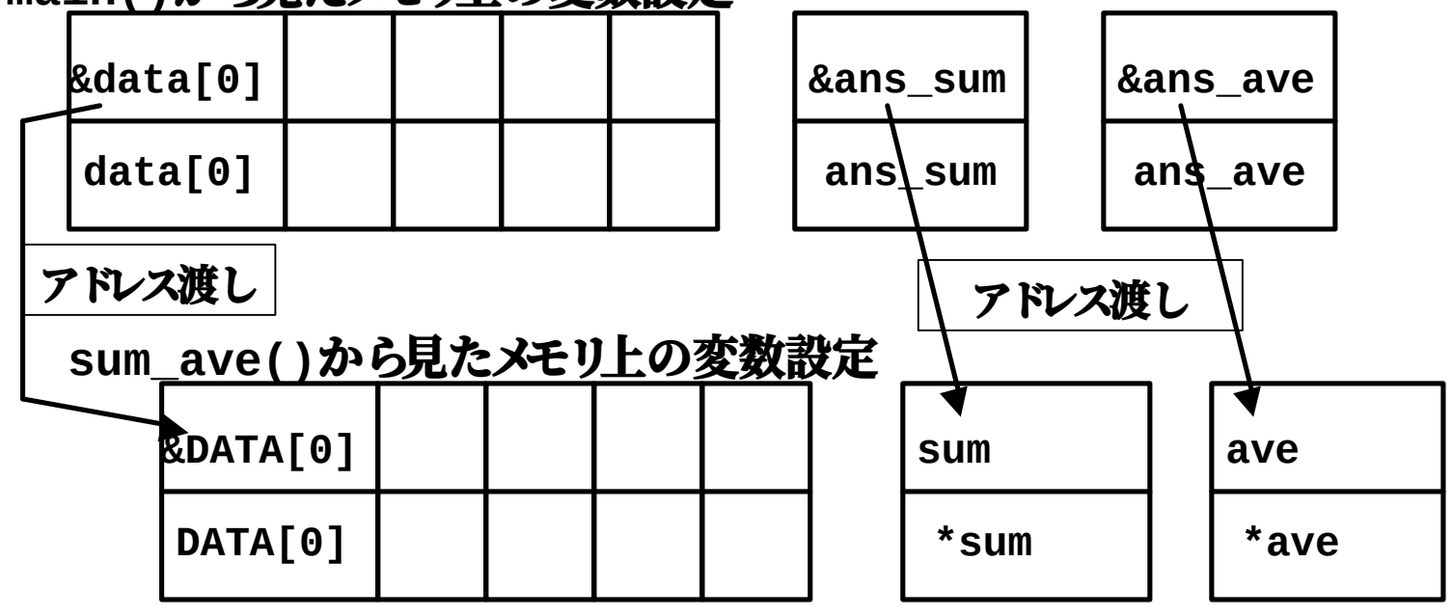
変数ans_sum, ans_aveのアドレスを引き渡す

関数**sum_ave()**内では、**ポインタ変数*sum,*ave**としていることに注意せよ

sum=&ans_sum;
ave=&ans_ave;
と同値となる

ポインタと関数２

main()から見たメモリ上の変数設定



- 変数のアドレスを引き渡すことで、異なる関数間で変数値を共有できる
- 配列を引き渡す場合、その先頭アドレス(**基底アドレス**)を渡すことで、各成分値の共有ができる
(data[0],data[1],･･･,data[4]→DATA[0],DATA[1],･･･,DATA[4])
- 配列の成分数(この場合、5ヶ)は、別個に引き渡す必要がある

(疑問点： 配列の基底アドレスを引き渡す先(仮引数)は、配列ではなく、**アドレス変数**でもよいのでは？
また、各成分は、アドレスを一つづつたどっていけば表現できるのでは?)

課題(1)

教科書Step24 p.99 sample24_1.cを、以下のように書き換えても
正しく計算されることを確認せよ

・配列の先頭アドレス(この場合data)を引き渡す先を、**ポインタ変数*DATA**としてみる

```
void sum_ave(int DATA[],int size,int *sum,float *ave)
↓
void sum_ave(int *DATA,int size,int *sum,float *ave)
```

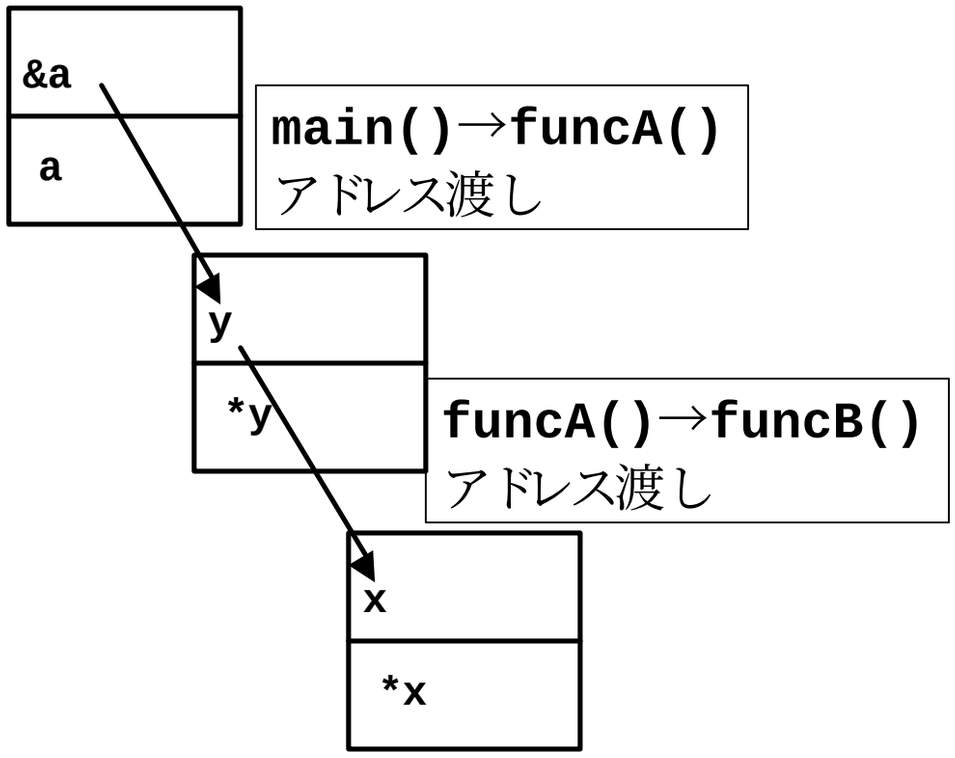
・関数sum_ave()内では、配列成分の表現DATA[i]を使わず、**間接演算子***と**ポインタ変数*DATA**を用いて表す

```
*sum+=DATA[i];
↓
*sum+=?????;
```

ヒント: 2006/12/22の講義(教科書Step23 p.95 sample23_1.c)を参照のこと
配列**a[5]**の先頭アドレス**a**をポインタ変数***p**に代入**(p=a;)**
↓
***(p+i)**と**a[i]**は同じとなる

ポインタと関数３（教科書Step24 p.100 sample24_2.c ）

```
#include <stdio.h>
void funcB(int *x){
        *x=2*(*x);
}
void funcA(int *y){
        *y=(*y)*(*y);
        funcB(y);
}
main()
{
        int a=10;
        funcA(&a);
        printf(”%d¥n”,a);
}
```



変数aのアドレス&a→ポインタ変数*y→ポインタ変数*xへと渡される

a*a*2の演算が実行され、答えは200となる

文字列１（教科書Step26 p.107 sample26_1.c ）

```
#include <stdio.h>
main()
{
        char a,b,c;
        a=65;
        b=0x42;
        c='C';
        printf("a = %c (%d, %x)¥n",a,a,a);
        printf("b = %c (%d, %x)¥n",b,b,b);
        printf("c = %c (%d, %x)¥n",c,c,c);
}
```

char型変数の定義

16進数で表す

シングルクウォテーション('')
で括られた場合、文字1つ
(本プログラムではC)を表す

アスキー(ASCII)コード表に従って文字A,B,Cを表示するプログラム
%c (文字表示) → A　B　C
%d (10進表示) → 65　66　67
%x (16進表示) → 41　42　43

実行例

```
a = A (65, 41)
b = B (66, 42)
c = C (67, 43)
```

文字列２（教科書Step26 p.107 sample26_2.c ）

```
#include <stdio.h>
main()
{
        char a[4] = "abc";
        printf("a[0] = %c (%d)¥n",a[0],a[0]);
        printf("a[1] = %c (%d)¥n",a[1],a[1]);
        printf("a[2] = %c (%d)¥n",a[2],a[2]);
        printf("a[3] = %c (%d)¥n",a[3],a[3]);
        printf("a = %s (%x)¥n",a,a);
}
```

文字列を表すために**char型(文字型)配列**を利用する

ダブルクウォテーション(**""**)で括られた文字列には、末尾に**ヌル(null)文字(¥0)**が付加されるため、少なくとも**文字数+1個の成分を持つ配列**を用意する必要がある

%s:　文字列表示)

文字型配列a[4]の基底アドレス

文字型配列a[4]のメモリ上の表現(末尾に**ヌル文字**付加)

&a[0]	&a[1]	&a[2]	&a[3]
a[0]=a	a[1]=b	a[2]=c	a[3]=¥0

実行例

```
a[0] = a (97)
a[1] = b (98)
a[2] = c (99)
a[3] =  (0)
a = abc (efbfdd48)
```

文字列３（教科書Step26 p.109 sample26_3.c ）

```
#include <stdio.h>
#include <string.h>
main()
{
        char str[10];
        strcpy(str, ”ABC”);
        printf(”str = %s¥n”,str);
        strcat(str, ”DEF”);
        printf(”str = %s¥n”,str);
        printf(”len = %d¥n”,strlen(str));
}
```

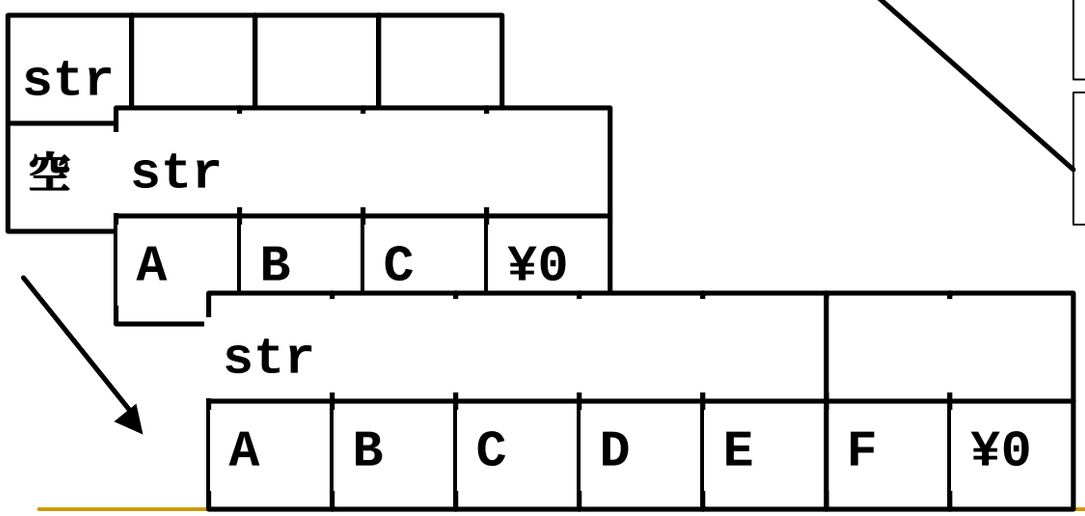
文字列操作関数
strcpy(),strcat(), strlen()を利用するため のヘッダー**string.h**

文字型配列str[10]

文字型配列str[10]の
基底アドレスを実引数にとり、
文字列ABC¥0をコピーする

str[10]内の文字列ABC¥0から、**ヌル文字(¥0)のみを消し**新たに文字列DEF¥0を追加する

str[10]内の文字数
(ヌル文字は除外する)を数える



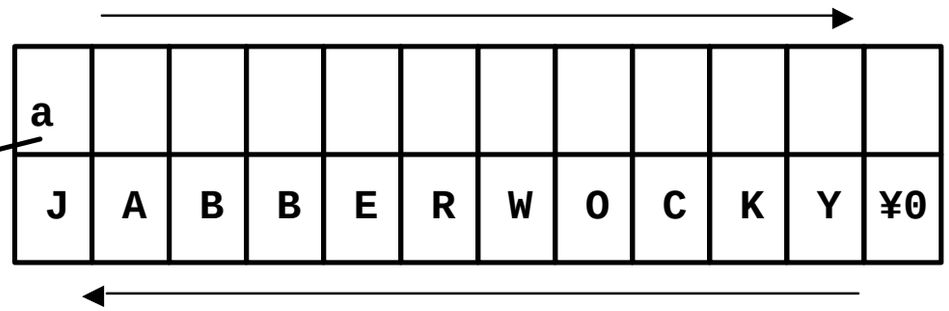
実行例

```
str     =       ABC
str     =       ABCDEF
len     =       6
```

文字列４（基底アドレスとヌル文字の利用例）

```
#include<stdio.h>
main()
{
char a[100] = ”JABBERWOCKY”;
char *a_ptr;
a_ptr = a;
        while(*(a_ptr) != ’¥0’)
        {
        printf(”%c ”,*(a_ptr++));
        }
        a_ptr--;
        while(a_ptr >= a)
        {
        printf(”%c ”,*(a_ptr--));
        }
printf(”¥n”);
}
```

配列a[100]の基底アドレス



基底アドレスから、ヌル文字(**¥0**)に達するまで文字を1つづつ表示

ヌル文字(**¥0**)の位置から一つアドレスを戻す(Yの位置)

Yの位置から、基底アドレスに達するまで文字を1つづつ表示

実行例

```
J A B B E R W O C K Y Y K C O W R E B B A J
```

(因みに、JABBERWOCKYとは 鏡の国のアリス」に出てくる、ちんぷんかんぷんな詩のタイトルのこと)

課題(2)（ファイルから文字列を読み込むプログラムの例）

```
#include <stdio.h>
main()
{
FILE *fp;
char fname[30] ="input.txt"; /*入力ファイル名はinput.txt*/
char a[100];
char *a_ptr;
a_ptr=a;
fp=fopen(fname,"r"); /*ファイルを読み込みモード"r"でオープン*/
fscanf(fp,"%s",a); /*ファイルからデータを一列読み込み、配列a[100]へ格納*/
fclose(fp);
        while(*a_ptr != '¥0')
        {
        printf("%c",*a_ptr);
        a_ptr++;
        }
printf("¥n");
}
```

文字配列fname[30]の基底アドレスを
fopen()へ引き渡していることに注意

あらかじめテキストファイルinput.txtを作り、ファイル中にJABBERWOCKYと書いておく。
画面にJABBERWOCKYと表示されることを確認せよ

課題(1) 解答例(変更部分を赤字表示)

```
#include <stdio.h>
void sum_ave(int *DATA,int size,int *sum,float *ave)
{
     int i;
     *sum=0;
     for(i=0;i<size;i++)
       {
        *sum+=*(DATA+i);
        }
     *ave=(float)*sum/(float)size;
}
main()
{
       int data[5]={1,2,3,4,5},ans_sum;
       float ans_ave;
       sum_ave(data,5,&ans_sum,&ans_ave);
printf(”sum=%d ave=%f¥n”,ans_sum,ans_ave);
}
```