

データ構造入門及び演習

4回目:ポインタ2

~文字列処理~

2014/05/02

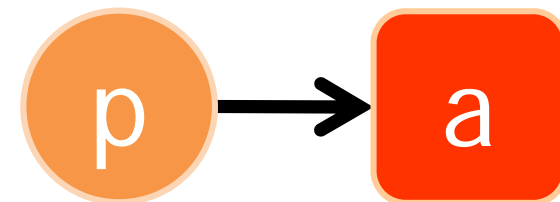
担当:見越 大樹

61号館304号室

ポインタ(復習)

- ポインタとは？
 - 変数が格納されている位置(アドレス)を値とする変数
- ポインタの宣言：
 - 型名の後ろ もしくは 変数名の前にアスタリスクをつける
char* p; または char *p;
(pをポインタ変数と呼ぶ)
- ポインタへのアドレスの代入：
 - ポインタpに変数aのアドレスを代入

```
char* p;  
char a='T';  
p = &a; (意味： p=aのアドレス)
```



pはaを指す
(pはaのアドレスを持つ)

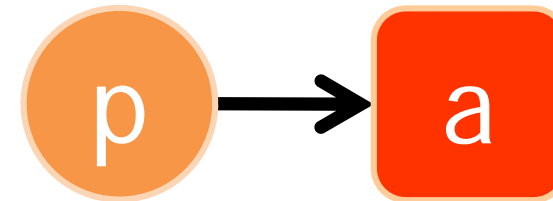
ポインタが指す値の参照(復習)

- ポインタ名の前に「*(アスタリスク)」をつけると、そのポインタが指す先の値を参照できる

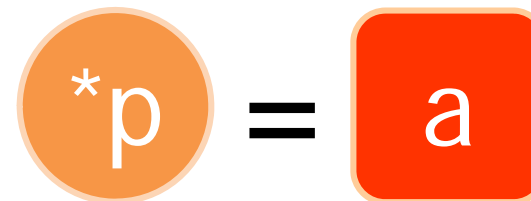
```
char* p;  
char a='T';  
p = &a; (意味：p=aのアドレス)
```

```
char b;  
b = *p; (意味：b='T')
```

- ポインタpを使って、変数aの値を変数bに代入





pはaを指す
(pはaのアドレスを持つ)





*pはaと同じ意味を持つ
*pはaのエイリアスと呼ぶ

注意：b=*pの「*」と、char* pの「*」は意味が異なる！

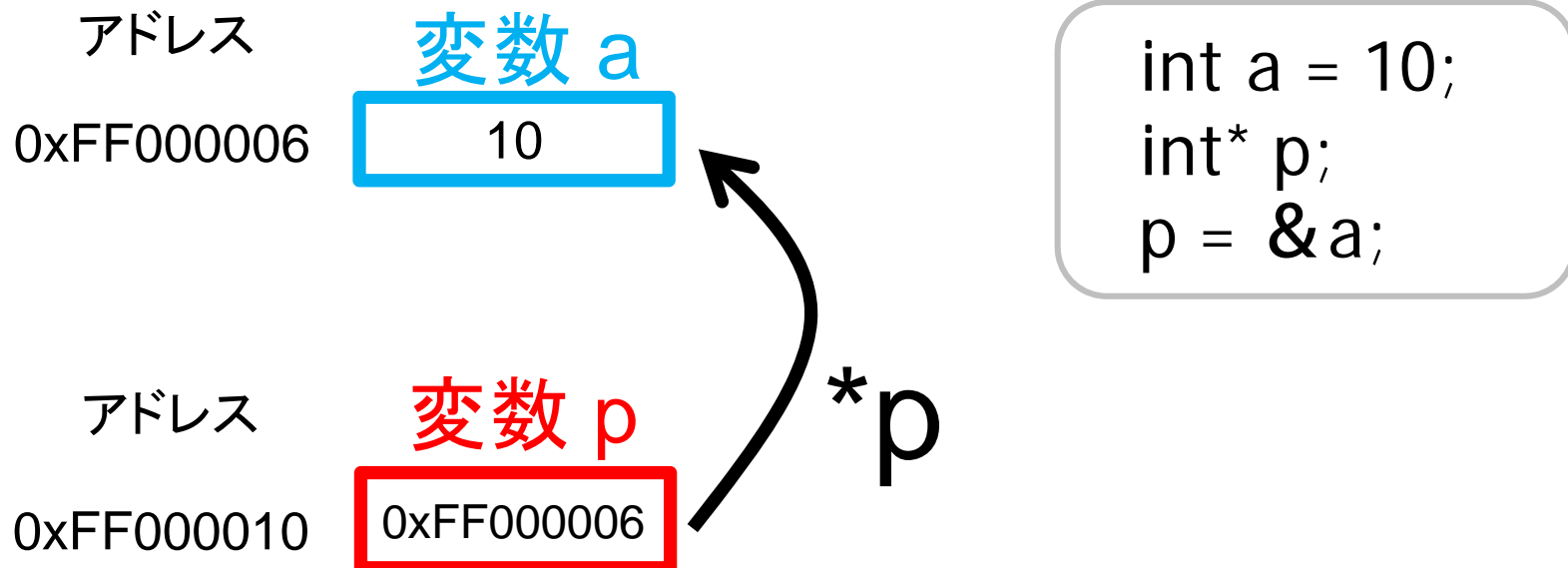
変数の型と格納できる値(復習)

アドレス	メモリ		
0xFF000006		← int a	変数aに整数型を格納できる
0xFF000010		← int* p	変数pにint 型変数のアドレスを格納できる

アドレス	メモリ	
0xFF000006	 10	← int a
0xFF000010	 0xFF000006	← int* p

```
int a = 10;  
int* p;  
p = &a;
```

ポインタの値とエイリアス

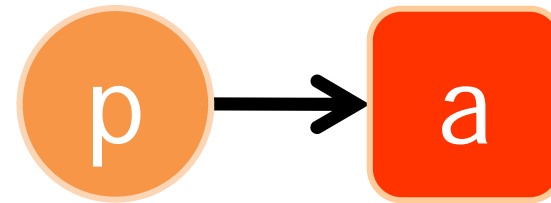


「p」にはアドレスが入っている

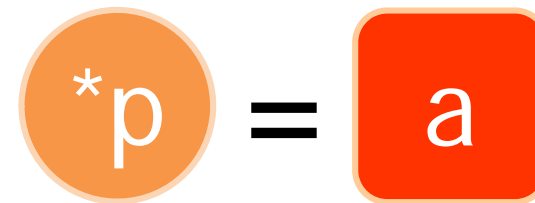
「*p」は変数pの指した先(アドレス)に飛ぶ
(*pとaは同じ意味を持つ)

ポインタのまとめ(復習)

```
int a = 10;  
int *p;  
p = &a;
```

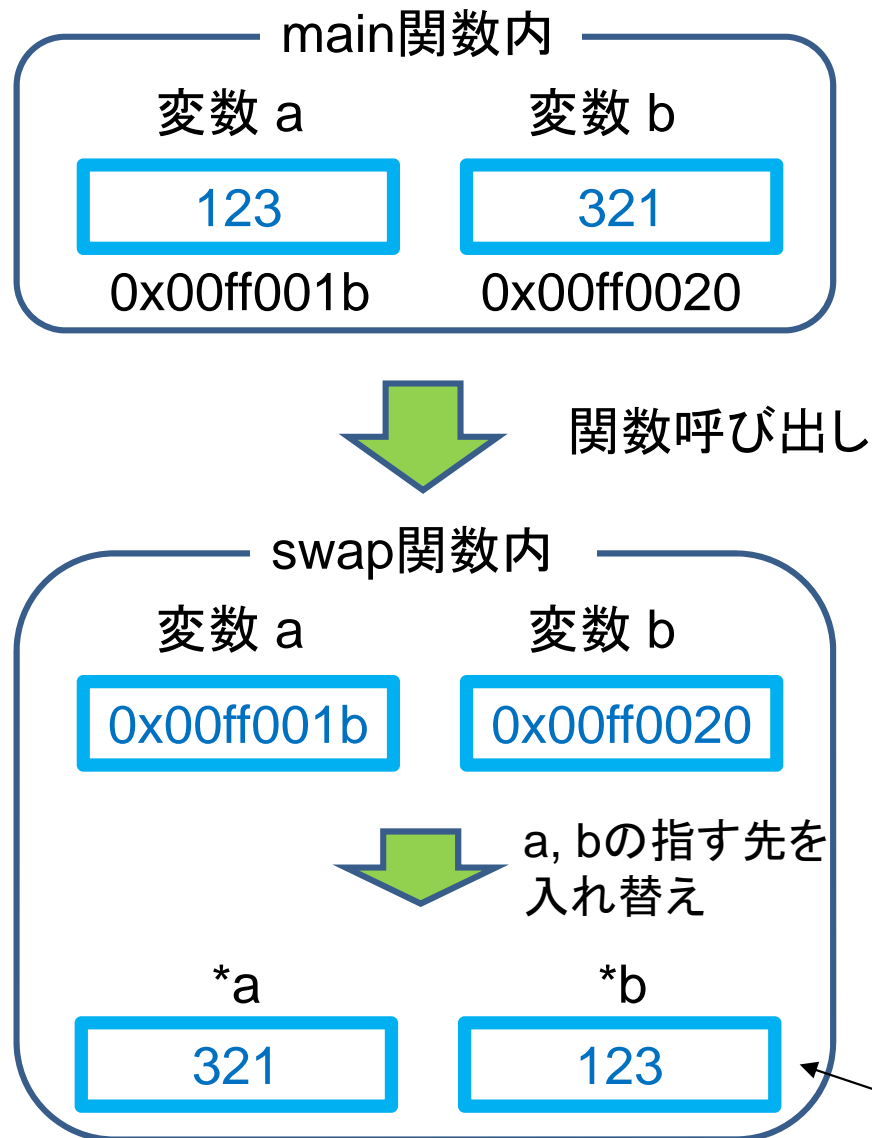


pはaを指す
(pはaのアドレスを持つ)



*pはaと同じ意味を持つ
*pはaのエイリアスと呼ぶ

関数の引数にポインタを使用する例(復習)



正しいプログラム

```
int main()
{
    int a=123,b=321;
    swap(&a, &b);
    . . .
}
```

```
void swap(int *a , int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```

これらのa,b
はアドレスを
通して結びつ
いている

main関数内の変数a,bはポインタを
使って書き換えられる

ポインタと配列(復習)

```
int a[4];
```

← aはa[0]へのポインタを表す

変数名	値	アドレス	ポインタ
a[0]	44	0x11ffab02	← a
a[1]	38	0x11ffab06	← a+1
a[2]	12	0x11ffab0a	← a+2
a[3]	25	0x11ffab0e	← a+3

int型の場合、ポインタが1加算されると、
アドレスは4ずつ加算される。

配列と文字列

- 配列とは？

- 同じ型のデータをまとめて格納するもの
- 宣言方法: `int x[4];`

整数型 名前 サイズ



x[0]	
x[1]	
x[2]	
x[3]	

- 文字列とは？

- 「文字型」のデータをまとめて格納するもの
- 宣言方法: `char m[6];`

整数型 名前 サイズ



m[0]	
m[1]	
m[2]	
m[3]	
m[4]	
m[5]	

文字列

- `char m[7];`と宣言したら,
 - 配列のサイズは「7」
 - 配列の添え字は「0から6」
- 文字列を格納する場所として実際に使えるのは,
 - 0から(サイズ-2)まで
 - 下の例の場合, 「0から5」まで
 - 文字列の末尾を表す特別な文字定数「ヌル文字(`'\0'`, 値は0)」を入れるために, 配列要素が1つ使われる

S	A	M	P	L	E	¥0
m[0]	m[1]	m[2]	m[3]	m[4]	m[5]	m[6]

文字列

- `char m[7];`と宣言したら,
 - 配列のサイズは「7」
 - 配列の添え字は「0から6」
- 文字列を格納する場所として実際に使えるのは,
 - 0から(サイズ-2)まで
 - 下の例の場合, 「0から5」まで
 - **文字列の末尾を表す**特別な文字定数「ヌル文字(`'\0'`, 値は0)」を入れるために, 配列要素が1つ使われる

H	E	L	L	O	¥0	
m[0]	m[1]	m[2]	m[3]	m[4]	m[5]	m[6]

文字列を配列に代入する方法

- 要素数を指定せずにchar型の配列を宣言し, 初期値として入力
`char m[] = "SAMPLE";`
- 要素数を指定してchar型の配列を宣言し, 各要素に1文字ずつ代入し, 末尾にヌル文字を入れる

`char m[7];`

`m[0] = 'S'; m[1] = 'A'; m[2] = 'M'; m[6]='¥n'`

S	A	M	P	L	E	¥0
---	---	---	---	---	---	----

m[0] m[1] m[2] m[3] m[4] m[5] m[6]

文字列を配列に代入する方法

- 要素数を指定してchar型の配列を宣言し, strcpyという標準関数で文字列を代入する (string.hをインクルードする)

```
char m[7];
```

```
strcpy(m, "SAMPLE");
```

S	A	M	P	L	E	¥0
m[0]	m[1]	m[2]	m[3]	m[4]	m[5]	m[6]

文字列の入出力

- 「書式」と「変数名」を書く

```
char m[7];  
scanf("%s", m);
```

書式 変数名

注) 変数名の前に & をつけない
m はポインタと同じ

```
printf("入力された文字列は%sです. ¥n", m);
```

書式 変数名

メモリの静的・動的な割り当て



- 静的な割り当て
 - `int temp; int x[5000]; char m[10000];`
 - 自動的にメモリ上に領域が確保される
 - プログラム中で「メモリを解放できない」

int x →

E000	E001	E002	E003	E004	E005	E006	E007
E010	E011	E012	E013	E014	E015	E016	E017
E020	E021	E022	E023	E024	E025	E026	E027
E030	E031	E032	E033	E034	E035	E036	E037
E040	E041	E042	E043	E044	E045	E046	E047
E050	E051	E052	E053	E054	E055	E056	E057
E060	E061	E062	E063	E064	E065	E066	E067
E070	E071	E072	E073	E074	E075	E076	E077

- 動的な割り当て
 - プログラム中で「メモリを解放できる」
 - 開放されたメモリは他のデータが使用可能になる
 - メモリを有効に利用できる
 - 割り当て方法: `malloc`関数を使用する!

E000	E001	E002	E003	E004	E005	E006	E007
E010	E011	E012	E013	E014	E015	E016	E017
E020	E021	E022	E023	E024	E025	E026	E027
E030	E031	E032	E033	E034	E035	E036	E037
E040	E041	E042	E043	E044	E045	E046	E047
E050	E051	E052	E053	E054	E055	E056	E057
E060	E061	E062	E063	E064	E065	E066	E067
E070	E071	E072	E073	E074	E075	E076	E077

E000	E001	E002	E003	E004	E005	E006	E007
E010	E011	E012	E013	E014	E015	E016	E017
E020	E021	E022	E023	E024	E025	E026	E027
E030	E031	E032	E033	E034	E035	E036	E037
E040	E041	E042	E043	E044	E045	E046	E047
E050	E051	E052	E053	E054	E055	E056	E057
E060	E061	E062	E063	E064	E065	E066	E067
E070	E071	E072	E073	E074	E075	E076	E077

← 解放

メモリの動的割り当て

- **malloc関数**

- 引数で指定された大きさ(バイト数)の領域を確保し, その領域の先頭ポインタを返す
- 使用例: `malloc(sizeof(int));`

- **free関数**

- 変数が記憶されている領域を解放し, 他の変数が使用できるようにする
- 使用例: `free(x)`

- **sizeof関数**

- 型名に割り当てられるバイト数を得る
- 使用例: `sizeof(int)`

メモリの動的割り当て

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

```
void main(void)
{
    char *m;
    m = (char*)malloc(7*sizeof(char));
    scanf("%s", m);
    printf("文字列は%sです. ¥n", m);
    free(m);
}
```

(1)ポインタを宣言

(2)メモリを確保し、先頭のアドレスを用意していたポインタに格納

// (1)

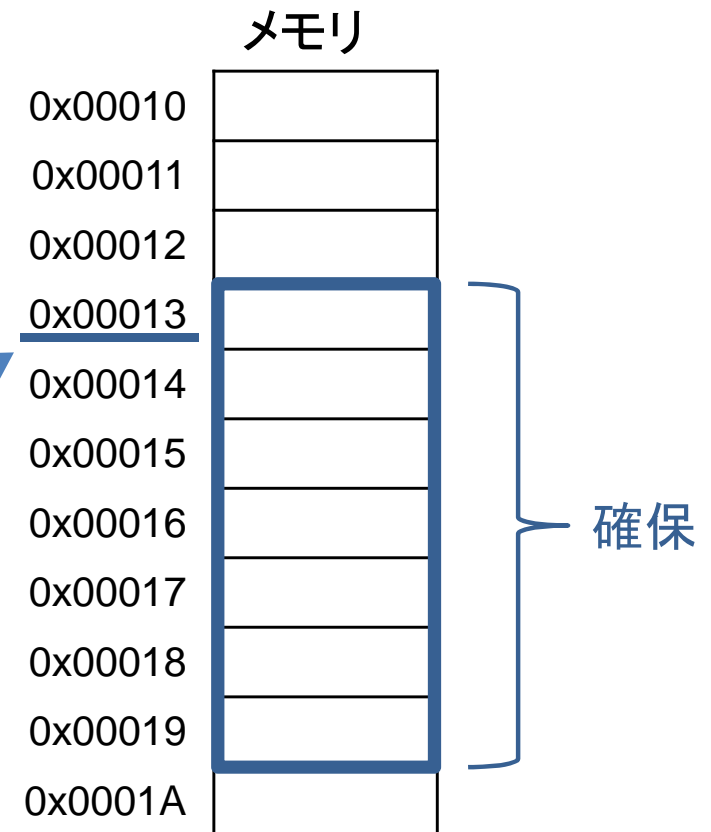
// (2)

// (3)

// (3)

// (4)

先頭のアドレス



(3)確保したら通常の配列と同様に使用可能

メモリの動的割り当て

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

```
void main(void)
```

```
{
```

```
    char *m; // (1)
```

```
    m = (char*)malloc(7*sizeof(char)); // (2)
```

```
    scanf("%s", m); // (3)
```

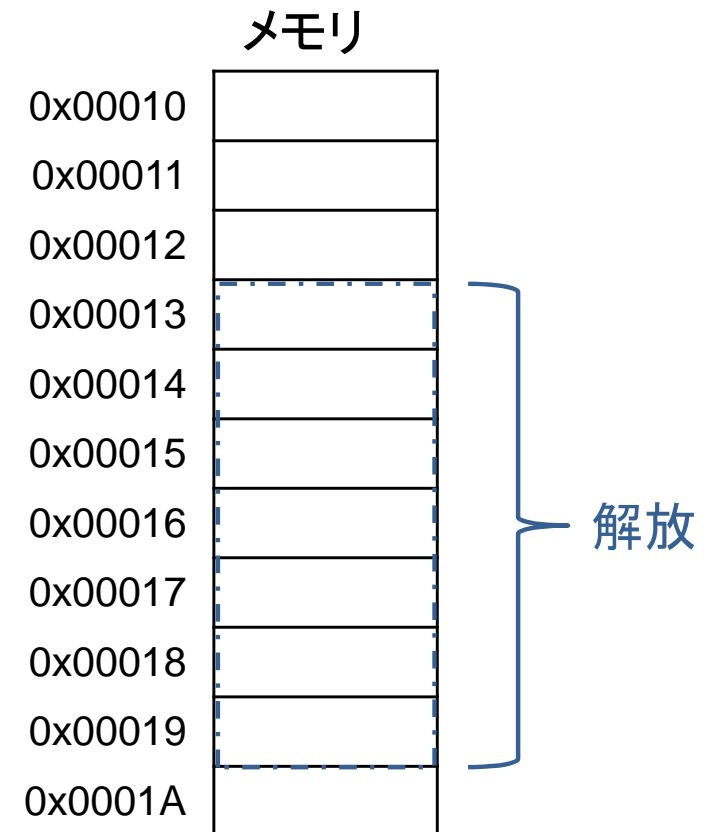
```
    printf("文字列は%sです. ¥n", m); // (3)
```

```
    free(m); // (4)
```

```
}
```

(1)ポインタを宣言

(2)メモリを確保し, 先頭のアドレスを用意していたポインタに格納



(3)確保したら通常の配列と同様に使用可能

(4)メモリの解放

文字列をポインタに代入する方法

- char型のポインタ変数を宣言し、初期値を入力する

```
char *m = "SAMPLE";
```

- char型のポインタ変数を宣言し、変数に代入する

```
char *m;
```

```
m = "SMAPLE";
```

- malloc関数でメモリを確保して、strcpy関数で文字列を代入

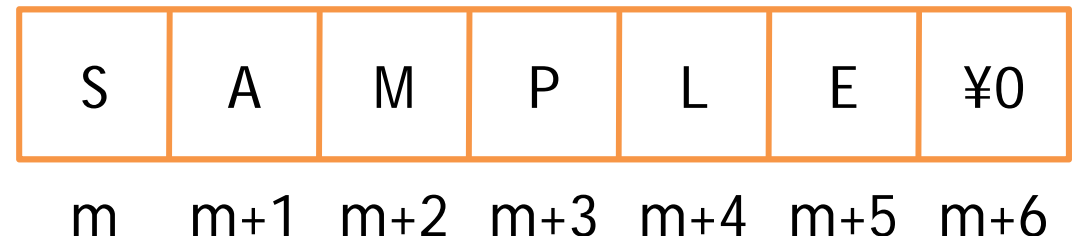
```
char *m;
```

```
m = (char*)malloc(7*sizeof(char));
```

```
strcpy(m, "SAMPLE");
```

```
...
```

```
free(m);
```



文字列処理におけるポインタの実行例

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    int i;
    char name[]="Nihon University";
    char *p;
    p=name;
    // name(アドレス)をポインタpに代入して確定する
    // pが指すアドレスが不定だと暴走する

    printf("*** case1 ***\n");
    printf("address: name=%p, p=%p\n", name, p);
    printf("value: name=%s, p=%s\n\n", name, p);

    printf("*** case2 ***\n");
    putchar(*p);          //putchar : 1 文字表示
    putchar(*(p+1));
    putchar(*(p+2));      //ポインタ(p+1)が指すアドレス
    putchar(*(p+3));      // の内容を表示
    putchar(*(p+4));
    putchar(*(p+5));
    putchar(*(p+6));
```

```
    putchar(*(p+7));
    putchar(*(p+8));
    printf("\n\n");
```

実行結果

```
*** case1 ***
address: name=0012FF40, p=0012FF40
(*コンピュータによって異なる)
value: name=Nihon University, p=Nihon
University

*** case2 ***
Nihon Uni
```

文字列処理における配列とポインタの実行例

- 配列を使用した場合

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

void main()
{
    int i;
    char name[]="Nihon University";

    char c;
    int num=0;

    printf("登録文字列は「%s」です¥n", name);
    printf("探す文字を入力してください:");
    scanf("%c",&c);
```

文字列の中から指定した1文字を見つけるプログラム

```
    i=0;
    while( name[i] != '¥0' ){
        if( name[i] == c ){
            printf("%d文字目に見つかりました！¥n",
                    i+1);
            num++;
        }
        i++;
    }

    if( num == 0 )
        printf("1つも見つかりませんでした・¥n");
    else
        printf("全部で%d個見つかりました・¥n", num);
}
```

文字列処理における配列とポインタの実行例

- ポインタを使用した場合

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

void main()
{
    int i;
    char name[]="Nihon University";
    char *p;
    p=name;
    // name(アドレス)をポインタpに代入
    char c;
    int num=0;

    printf("登録文字列は「%s」です\n", name);
    printf("探す文字を入力してください:");
    scanf("%c",&c);
}
```

文字列の中から指定した1文字を見つけるプログラム

```
while( *p != '¥0' ){
    if( *p == c ){
        printf("%d文字目に見つかりました！\n",
                p-name+1);
        num++;
    }
    p++;
}

if( num == 0 )
    printf("1つも見つかりませんでした・\n");
else
    printf("全部で%d個見つかりました・\n", num);
}
```

実行結果

```
登録文字列は「Nihon University」です  
探す文字を入力してください:i  
2文字目に見つかりました！  
9文字目に見つかりました！  
14文字目に見つかりました！  
全部で3個見つかりました.
```

文字数のカウント ~配列~

```
#include <stdio.h>
int fstrlen(char moji[]);

int main(void)
{
    char s1[80];
    int n;
    printf("s1=");
    scanf("%s",s1);
    n=fstrlen(s1);
    //引数として配列名(ポインタ)を渡す
    printf("%d¥n", n);
    return 0;
}
```

```
int fstrlen(char moji[])
{
    }
}
```

練習問題：
文字数をカウントする関数fstrlenを完成させなさい

文字数のカウント ~ポインタ~

```
#include <stdio.h>
int fstrlen(char *moji);
```

```
int main(void)
{
    char s1[80];
    int n;
    printf("s1=");
    scanf("%s",s1);
    n=fstrlen(s1);
    printf("%d¥n",n);
    return 0;
}
```

```
int fstrlen(char *moji)
{
    }
}
```

練習問題：
文字数をカウントする関数fstrlenを完成させなさい。

文字列の操作

- 標準関数を使用して、多様な操作ができる
 - 文字列処理用の標準関数を使用するときは、次の1行をプログラムに含めること
 - `#include <string.h>` (string: 文字列)
- 関数の例:
 - コピー: `strcpy(s, ct)` (copy: コピーする)
 - 長さ取得: `strlen(cs)` (length: 長さ)
 - 連結: `strcat(s, ct)` (catenate: 連結する)
 - 比較: `strcmp(cs, ct)` (compare: 比較する)

sprintf関数

- `#include <stdio.h> //sprintfを呼び出すためにincludeする`
`sprintf(str, “.....”,);`
- 書式formatにしたがって、printf関数と同様の変換を行った出力を、文字列strに格納
- 例)
`int n = 10;`
`char str[256]; //出力するための領域を確保`
`sprintf(str, “変数nの値は%nです”, n);`
`//確保した領域より文字列が長いとバグになる`

文字列操作関数の使用例

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

void main()
{
    char* str1 = "Hello";
    char* str2 = "World!";
    char str3[256] = {};
    int tmp = 0;

    tmp = strcmp(str1, str1);
    printf("%sと%sの比較結果 : %d\n\n",
           str1, str1, tmp);

    tmp = strcmp(str2, str2);
    printf("%sと%sの比較結果 : %d\n\n",
           str2, str2, tmp);

    tmp = strcmp(str1, str2);
    printf("%sと%sの比較結果 : %d\n\n",
           str1, str2, tmp);
```

```
    tmp = strlen(str1);
    printf("%sの長さは : %d\n\n", str1, tmp);

    strcpy(str3, str1);
    printf("strcpy(str3, str1)実行後のstr3は : %s\n\n",
           str3);

    strcat(str3, str2);
    printf("strcat(str3, str2)実行後のstr3は : %s\n\n",
           str3);

    sprintf(str3, "%s %s\n", str1, str2);
    printf("sprintf(str3, .... )実行後のstr3は : %s\n",
           str3);
}
```

実行結果

HelloとHelloの比較結果 : 0

World!とWorld!の比較結果 : 0

HelloとWorld!の比較結果 : -15

Helloの長さは : 5

strcpy(str3, str1)実行後のstr3は : Hello

strcat(str3, str2)実行後のstr3は : HelloWorld!

sprintf(str3,)実行後のstr3は : Hello World!