第4週
ポインタ引数と文字列処理

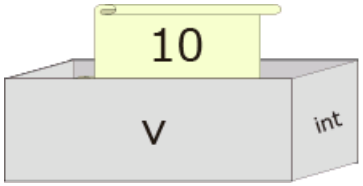
今週習得すること

- 関数のプロトタイプ宣言
- 値渡し
- ポインタ渡し
- ポインタによる文字列渡し
- 文字列操作<string.h>
  - strlen
  - strcpy
  - strcat
  - strcmp

値渡し

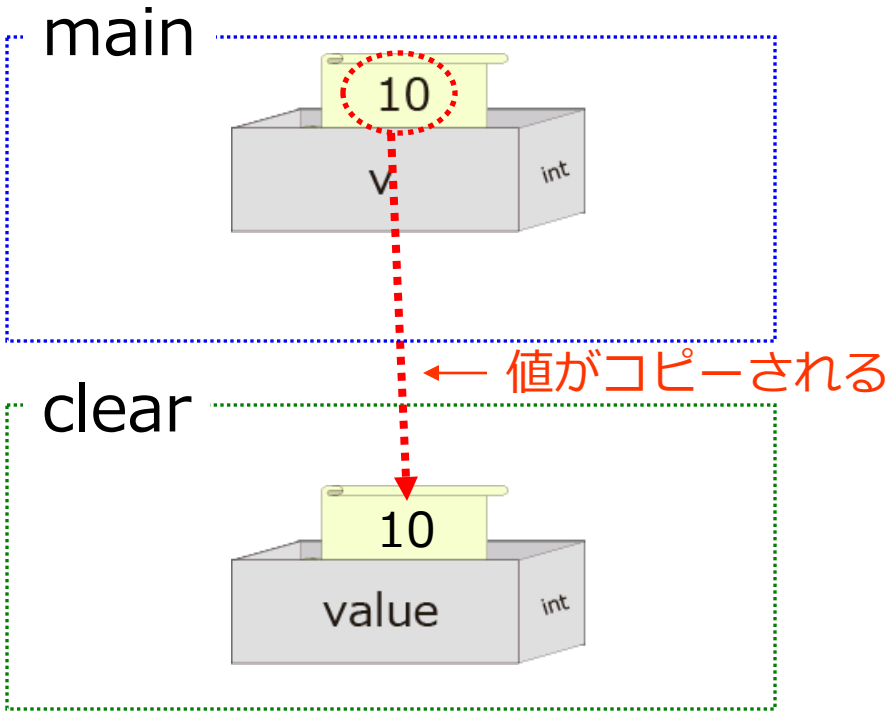
```
void clear( int value );

int main(){
  int v = 10;
  clear( v );
  printf("v = %d¥n", v );
  return 0;
}

void clear( int value ){
  value = 0;
};
```

関数のプロトタイプ宣言
プログラム内で利用する関数の
名前、引数、型を指定する文法規則



値渡し

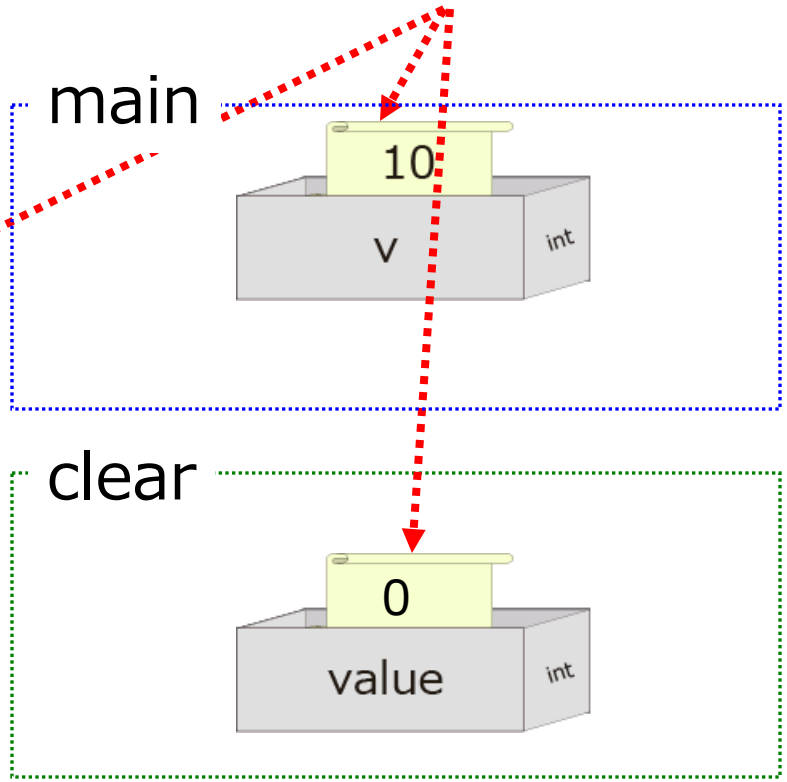
```
void clear( int value );

int main(){
  int v = 10;
  clear( v );
  printf("v = %d¥n", v );
  return 0;
}

void clear( int value ){
  value = 0;
};
```



値渡し

```
void clear( int value );

int main(){
  int v = 10;
  clear( v );
  printf("v = %d¥n", v );
  return 0;
}

void clear( int value ){
  value = 0;
};
```

valueに値が代入されても
vとvalueは別物なので
vに影響しない

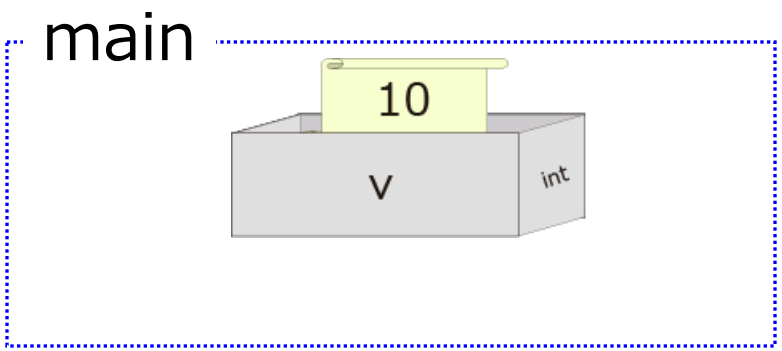


値渡し

```
void clear( int value );

int main(){
  int v = 10;
  clear( v );
  printf("v = %d¥n", v );
  return 0;
}

void clear( int value ){
  value = 0;
};
```

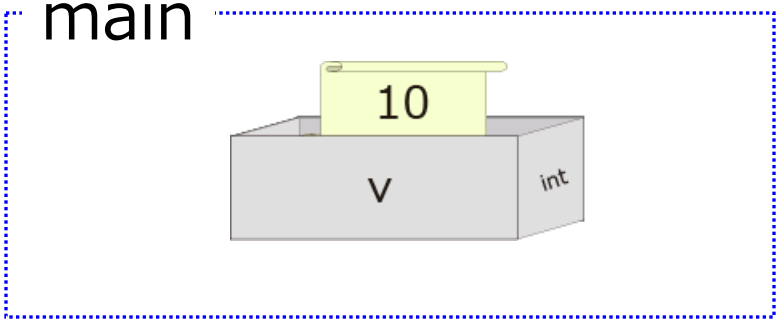
出力結果

```
v = 10
```



ポインタ渡し

```
void clear( int *value );

int main(){
  int v = 10;
  clear( &v );
  printf("v = %d¥n", v );
  return 0;
}

void clear(int *value ){
  *value = 0;
};
```

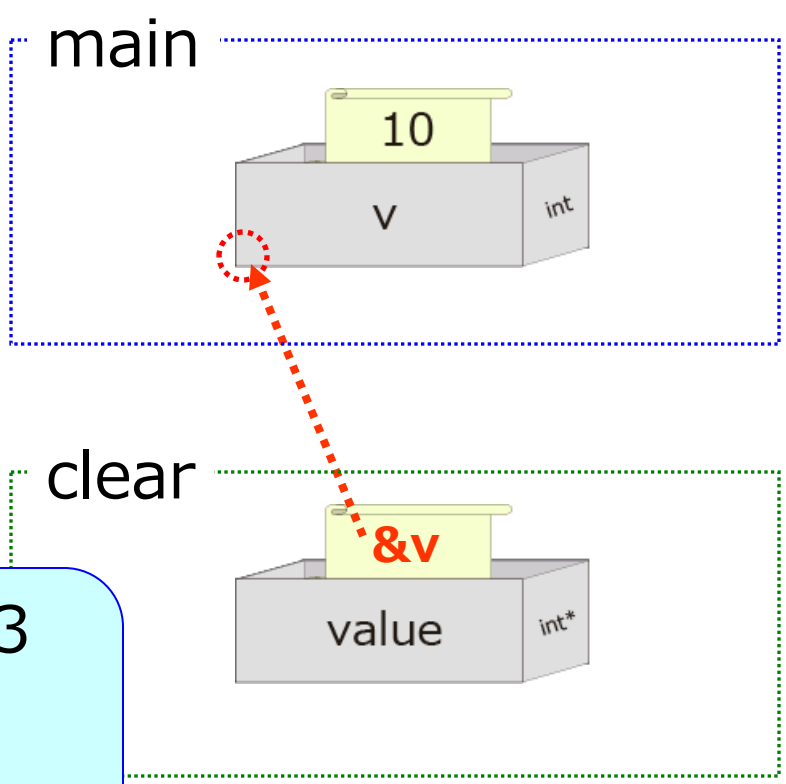


ポインタ渡し

２週目のポイント２
変数の前に**&**をつけると
変数のアドレスを取得できる

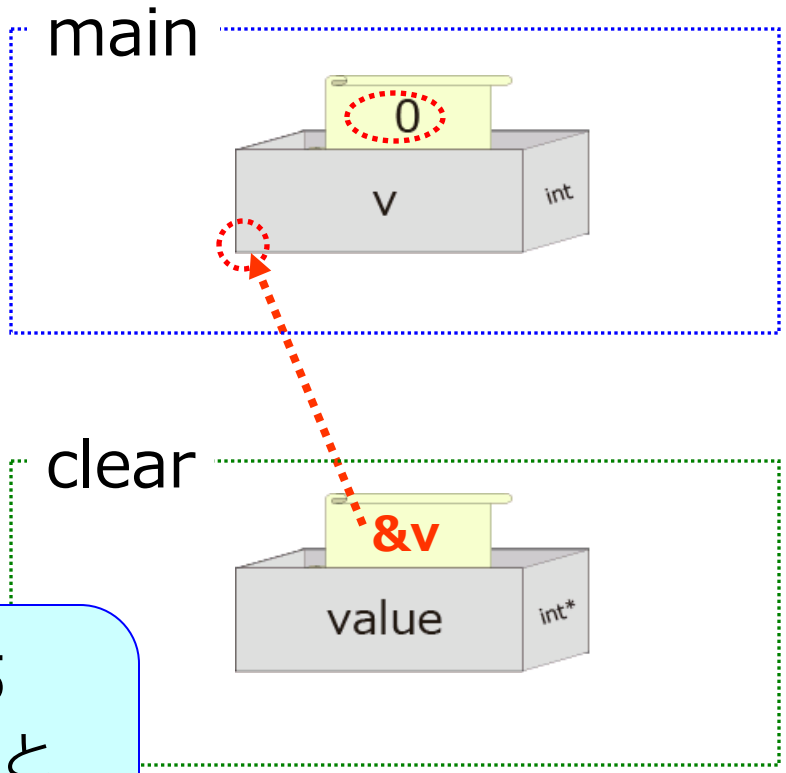
```
int main(){
  int v = 10;
  clear( &v );
  printf("v = %d¥n", v );
  return 0;
}

void clear(int *value){
  *value = 0;
};
```

２週目のポイント３
ポインタ変数の宣言は
型名 ＊ ポインタ変数名;



ポインタ渡し

```
void clear( int *value );

int main(){
  int v = 10;
  clear( &v );
  printf("v = %d¥n", v );
  return 0;
}

void clear(int *value ){
  *value = 0;
};
```



２週目のポイント５

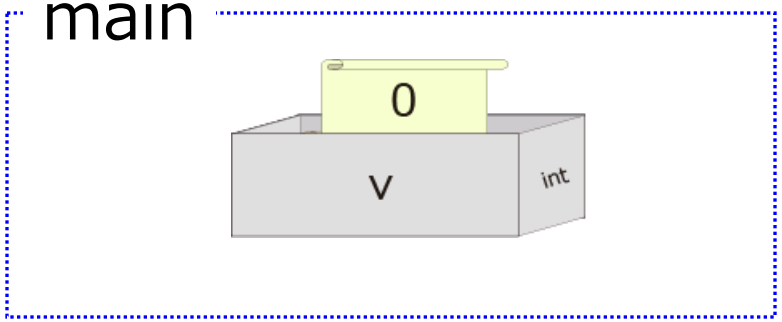
ポインタ変数に＊をつけるとポインタ変数の指している先の変数の値を参照できる

ポインタ渡し

```
void clear( int *value );

int main(){
  int v = 10;
  clear( &v );
  printf("v = %d¥n", v );
  return 0;
}

void clear(int *value ){
  *value = 0;
};
```

出力結果

```
v = 0
```



関数 -復習-

●入力（引数）に対しまとまった処理を行い結果を出力

```
#include <stdio.h>
void swap(int x, int y);

int main(){
  int x,y;
  x = 2; y = 5;
  swap(x,y);
  printf("x=%d, y=%d¥n",x,y);
}

void swap(int x, int y){
  int temp = x;
  x = y;
  y = temp;
}
```

プロトタイプ宣言

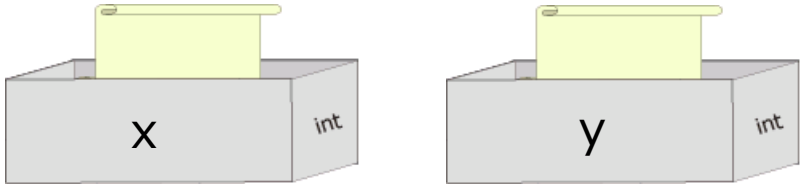
関数型（戻り値の型を指定）
return で戻り値を返す

仮引数

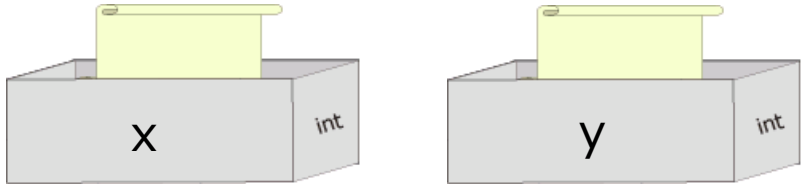
関数内の変数宣言

必須課題4－1:理解補助シート

main

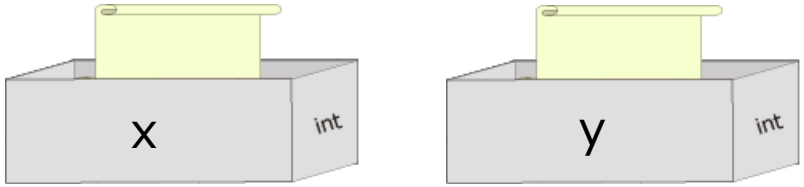


swap

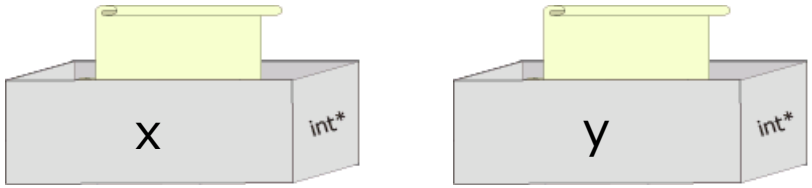


必須課題4－2:理解補助シート

main



swap



C標準ライブラリ

- 汎用性の高いプログラムをまとめたもの
  - ヘッダファイルをインクルードして利用

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

  - 入出力に便利なプログラムをまとめたライブラリ
    - printf, scanf, fgetsなどが利用できる

文字列操作のライブラリ string.h

- char str1[20], str2[20]
- **strlen**：文字列長を返す

```
strlen( str1 );
```

str1の長さを返す(¥0は含めない)

- **strcpy**：文字列のコピー

```
strcpy( str1, str2 );
```

str2の内容をstr1へコピー

- **strcat**：文字列の連結

```
strcat( str1, str2 );
```

str1にstr2を連結する

- **strcmp**：文字列の比較

```
strcmp( str1 );
```

str1 = str2 ⇒ 0
str1 > str2 ⇒ 0より大きい値
str1 < str2 ⇒ 0より小さい値

strlen:文字列のサイズを返す

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

int main(){
  char str[10] ="ABCD";
  printf("str size is %d¥n", strlen(str));
  return 0;
}
```

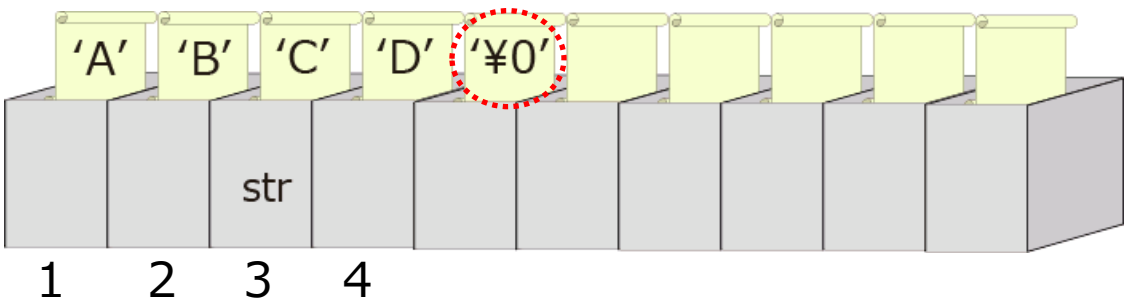
文字列操作の関数（strlen,strcpy, strcat, strcmpなど）を利用するには string.hをインクルードすること

size_t strlen(**char *s**);
返り値:文字列の長さ

実行結果

```
str size is 4
```

¥0はカウントされない



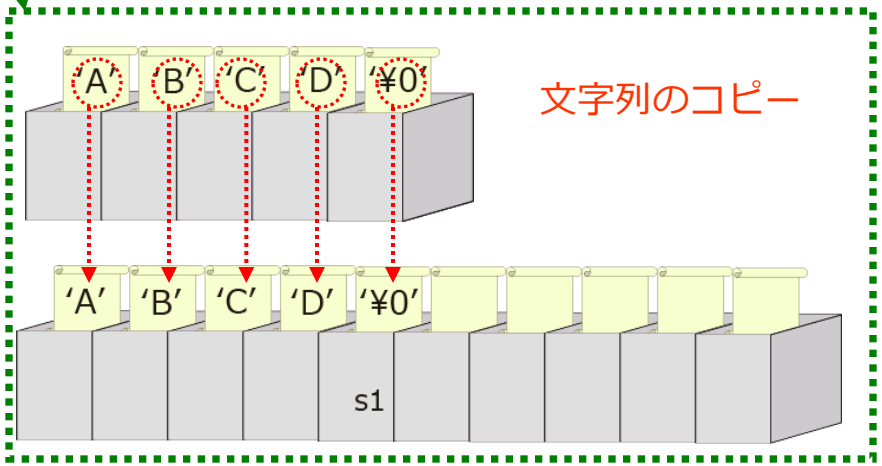
strcpy:文字列のコピー

strcpy(**char *s1, char*s2**);
文字列**s2**を**s1**の領域にコピーする
注意：s1がs2よりサイズが小さいと
データを壊す可能性あり

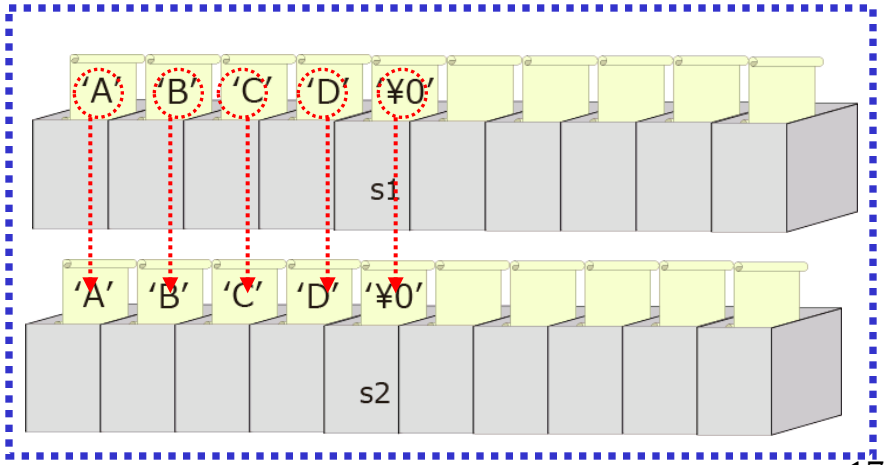
```
int main(){
  char s1[10];
  char s2[10];
  strcpy(s1, "ABCD");//①
  strcpy(s2, s1);      //②
  printf("s1 = %s  s2 = %s¥n",s1, s2);
  return 0;
}
```

実行結果　s1 = ABCD s2 = ABCD

① strcpy(**s1**, **"ABCD"**);



② strcpy(**s2**, **s1**);



strcat:文字列の連結

```
int main(){
  char s1[10] = “ABCD”;
  char s2[5] = “DEFG”;
  strcat(s1, s2);
  printf(“s1 = %s  s2 = %s¥n”,s1, s2);
  return 0;
}
```

strcat(**char *s1, char*s2**);

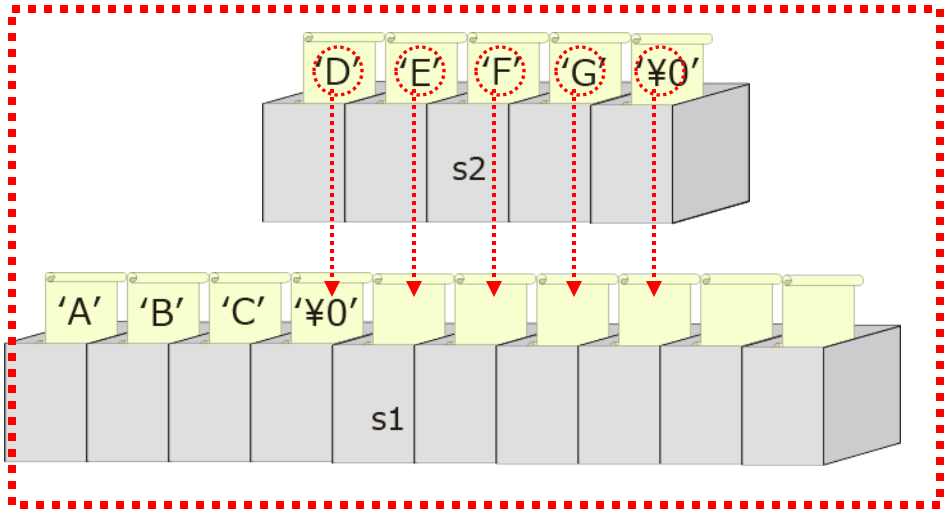
文字列**s2**を**s1**の末尾連結する

注意：s1のサイズが小さいと
データを壊す可能性あり

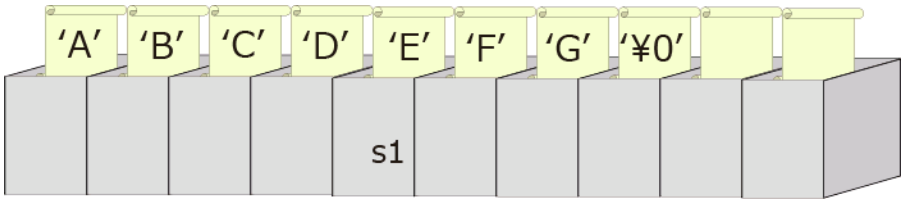
実行結果

```
s1 = ABCDDEFG  s2 = DEFG
```

strcat(**s1**, **s2**);



連結後の文字列



strcmp:文字列の比較

int strcmp(**char *s1, char*s2**);

s1と**s2**の文字列を比較

返り値：

0 ：**s1**と**s2**が同じ

正の整数：**s1**の方が大　(**s1**が**s2**より辞書順で後)

負の整数：**s1**の方が小　(**s1**が**s2**より辞書順で先)

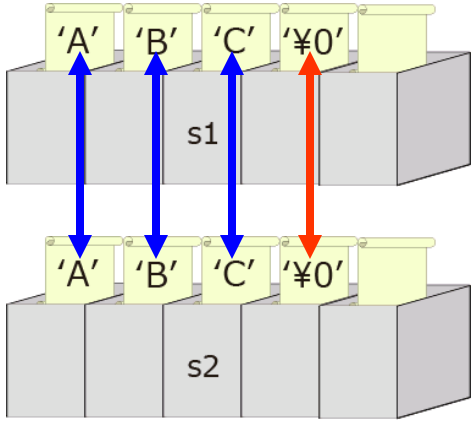
```
int main(){
  char s1[4] = "ABC";
  char s2[4] = "ABC";
  char s3[5] = "ABB";
  char s4[5] = "ABCD";

  printf("strcmp(s1, s2) = %d¥n", strcmp(s1, s2) );
  printf("strcmp(s1, s3) = %d¥n", strcmp(s1, s3) );
  printf("strcmp(s1, s4) = %d¥n", strcmp(s1, s4) );
  return 0;
}
```

値は処理系依存

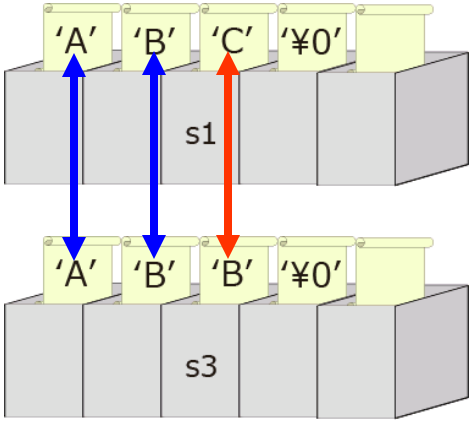
実行結果

```
strcmp(s1, s2) = 0
strcmp(s1, s3) = 1
strcmp(s1, s4) = -68
```

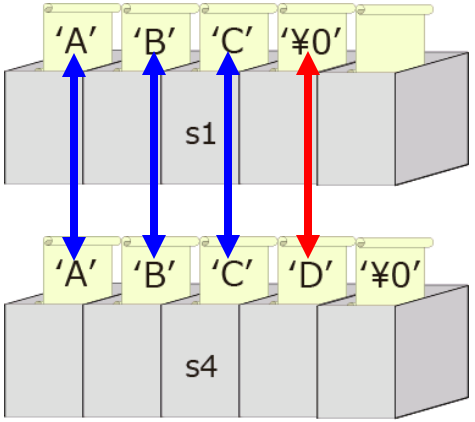
‘¥0’ – ‘¥0’ = 0



‘C’ – ‘B’ = 1



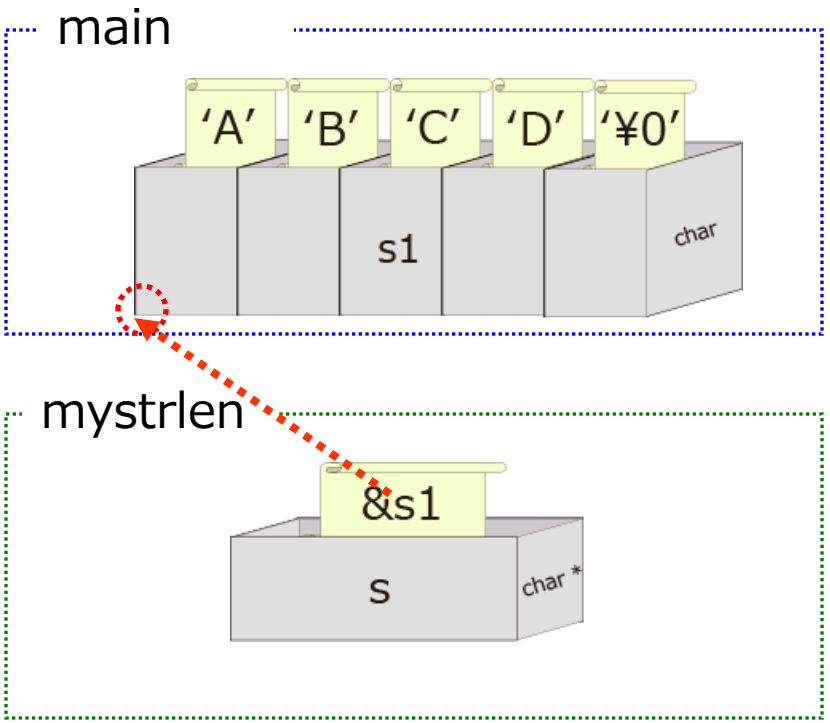
‘¥0’ – ‘D’ = -68



ポインタによる文字列渡し

```
#include<stdio.h>
size_t mystrlen(char *s);

int main(){
  char s1[5] = "ABCD";
  int len;
  len = mystrlen( s1 );
  printf("len = %d¥n", len);
  return 0;
}

size_t mystrlen( char *s ){
  size_t len = 0;
  while( *s != '¥0' ){
        len++;
        s++;
  }
  return len;
}
```

ポインタを利用することで
関数の引数に文字列を渡すことができる



必須課題4-3：作成手順

1. create_word_pair(char *a, char *b)
   レジュメのp10の指示通りに、構造体のメンバを設定する

2. 標準出力に１で作成した構造体のメンバをすべて出力する

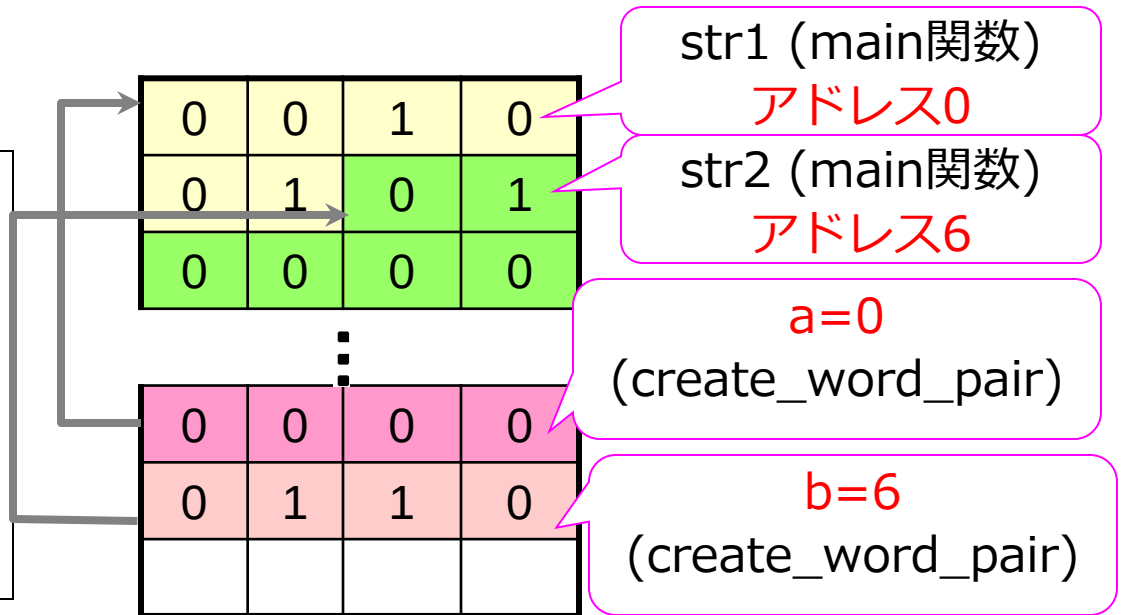
必須課題4-3 ポインタによる文字列渡し

- 以下の関数を作成
  - word_pair_t create_word_pair(char *a, char *b)
- main関数から以下のように呼び出し
  - char str1[SIZE1], str2[SIZE2]; /* SIZE1,2は適当に決めていい */
  - w = create_word_pair(str1, str2);

- 関数create_word_pair()はアドレスstr1, str2 (char型配列str1, str2の先頭のアドレス) を受け取る
- 受け取った先頭アドレスを用いて文字列にアクセス
- main関数側でも値が変わる

create_word_pair関数からのアクセス

- 知っているのはstr1,2の先頭のアドレスのみ．配列サイズを知らない
- 一般に，ヌル文字'¥0'が初出する位置で文字列の終端を判定



著者リスト

1. 安積 卓也（情報システム学科）
2. 泉　朋子 （情報コミュニケーション学科）
3. 原田 史子（情報システム学科）