プログラミング2　第3回

EX-2.2について

- ルールを書くのを忘れていた
  - 3*3のマスに〇, ×の順番で印を書く。
  - パスは不可。（よって最大9手で終わる）
  - 縦・横・斜めのいずれかで同じ印を３つ並べたら勝利。
    - 勝敗が決まったらそこで処理を終了する
  - マスが全部埋まっても、同じ印が3つ並ばなかったら引き分け。
- 勝敗の判定には斜めがあるので注意
- printf関数で文字を出力する際の指定子は%c（%sではない）

講義計画（暫定）

- ~~第1回(11/28(金))　ガイダンス、Cプログラミングの基礎（データ型、変数、四則演算、制御構文）~~
- ~~第2回(12/05(金))　ビット演算、配列、関数~~
- 第3回(12/12(金))　配列と関数、文字列、ソート
- 第4回(12/19(金))　再帰関数、ポインタ(1)
- 第5回(12/26(金))　ポインタ(2)
- 第6回(01/ 9(金))　構造体・共用体、リスト
- 第7回(01/23(金))　Cプログラミング実用上必須のもの（defineマクロ、ファイル操作、複数ソースコンパイルなど）
- 第8回(01/30(金))　期末試験（**出席必須！**）

関数への配列の渡し方

- EX-2.2では、グローバル領域に定義された二次元配列を使用した
  - グローバル変数は、プログラムのどこでも利用できるため、関数の引数に書く必要がなくて便利
  - しかし大規模プログラムや並列プログラムでは使うべきではない
    - 関数間にグローバル変数を介した依存関係が生じるため、どこかの関数を変更すると別の関数がバグる可能性がある

```
double val = 10.0;

double func1(){
    double ret = 0.0;
    if(val != 0){
        ret = 1.0 / val;
    }
    return ret;
}
```

```
void func2(){
    val = 0.0;
}
```

func1とfunc2を並列に実行すると、if文の後にvalの更新が行われ、ゼロ割りが発生する可能性がある

関数への配列の渡し方

- 再確認：配列について
  - 全てが同じ型を持ち、共通の名前によってアクセスできる変数のリスト（並び）のこと
  - 宣言の例：int型で20個の要素を持つ配列

```
int ary[20];
```

  - 配列名, インデックス番号, 大括弧で配列要素にアクセス ary[m]
  - 配列名だけの場合は配列のメモリにおける先頭**アドレス**を表す
- 変数、配列、ポインタの比較

`int x;` 整数の値を保存する。a = x + 1; のように使えば、xに保存された整数の値を使って計算する。&xとすると、xが使っているメモリのアドレスを取得できる。

`int y[10];` 整数の値を保存する変数のリスト。a = y[0] + 1; のように使えば、y[0]に保存された整数の値を使って計算する。&y[0]と書くと、y[0]が使っているメモリのアドレスを取得できる。単にyと書くと、配列yの先頭アドレス（つまり&y[0]）を取得できる。

`int *z;` 整数型の変数や配列のメモリアドレスをzに保存する。z = &y[0]; a = z[0] + 1; のように使えば、y[0]に保存された整数の値を使って計算する。

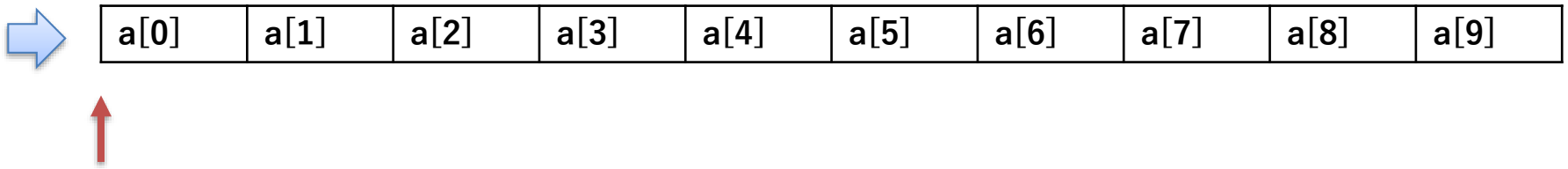
つまり配列は、サイズ情報を付加されたポインタのように振る舞う。詳細は次回以降。

関数への配列の渡し方

- 1次元配列の宣言

メモリ上でのイメージ

int a[10];

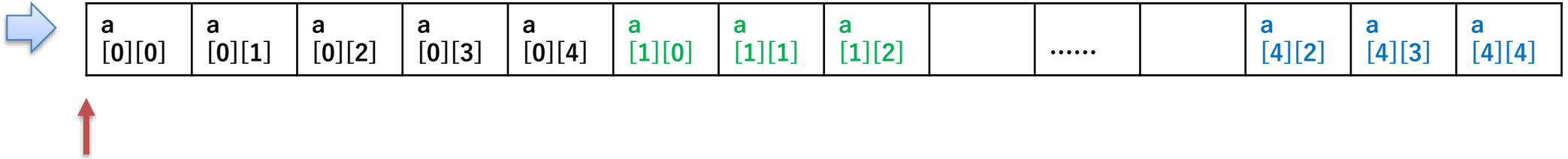


ここの先頭場所(アドレス)は配列名 **a** で表される
→ **a** は多くの場合、先頭要素へのポインタに自動変換される

- 2次元配列の宣言

メモリ上でのイメージ

int a[5][5];



ここの先頭場所(アドレス)は配列名 **a** で表される
→ **a** は多くの場合、最初の行へのポインタに自動変換される

1次元配列の仮引数指定方法

- 関数呼び出し時の実引数リストでの指定方法

```
double a[10];
func(10, a);
```

  - 配列を配列として（含まれる要素1つではなく配列として）渡すには、その配列がメモリ上のどこにあるか（場所）を渡す必要がある

- 関数本体での仮引数指定方法
  - 1. 配列の宣言と同様に指定

```
void func(int n, double a[10]) { … }
```

先頭場所の情報だけでは配列のサイズがわからないため、配列サイズも引数として渡す

  - 2. サイズは空白として配列であることだけを指定

```
void func(int n, double a[]) { … }
```

  - 3. ポインタ変数として指定

```
void func(int n, double *a) { … }
```

これらの書き方でも何らかの方法でnを伝えないと関数内部で配列長がわからず困ることになる点に注意（int nをa[]や*aより前方に書かねばならないわけではない）

  - 4. 可変長配列(Variable Length Array)として指定(C99の機能)

```
void func(int n, double a[n]) { … }
```

この書き方ではa[n]よりも前方にint nが必要

2次元配列の仮引数指定方法

- 関数呼び出し時の実引数リストでの指定方法

```
double a[4][3];
func(4, 3, a);
```

  - 2次元配列でもメモリ上での場所を渡す必要があり、
    さらに2次元形状が分かるように伝えなければならない

- 関数本体での仮引数指定方法
  - 1. 配列の宣言と同様に指定

```
void func(int m, int n, double a[4][3]) { ... }
```

  - 2. 行サイズは空白として2次元配列であることを指定

```
void func(int m, int n, double a[][3]) { ... }
```

  - 3. 3要素の配列を指すポインタとして指定(アスタリスクと括弧の付け方に注意)

```
void func(int m, int n, double (*a)[3]) { ... }
```

  - 4. 列サイズを可変長配列として指定(C99の機能)

```
void func(int m, int n, double a[][n]) { ... }
```

C言語における文字の扱い

- 文字にはchar型を使う
- char型は整数を扱う型でもある
- どういうことなのか？

```
char ch = 'C';
```

```
char n = 67;
```

- C言語における文字はASCIIコードに対応している
  - ASCIIコード：文字に数字を割り当てた標準的な対応表、アルファベットと記号に対応
  - char型変数1つにASCIIコード1文字を格納し、printfなどでは対応する文字を表示している
  - ASCIIコードと対応する文字の関係（の一部抜粋）

コード番号（10進）		35	36	37	38		48	49	50	51
コード番号（16進）		23	24	25	26		30	31	32	33
文字		#	$	%	&		0	1	2	3
コード番号（10進）		65	66	67	68		97	98	99	100
コード番号（16進）		41	42	43	44		61	62	63	64
文字		A	B	C	D		a	b	c	d

※ 0から31までは制御文字のため表示できない。出力して確認する際には注意。

文字列

- 文字列にはchar型配列を使う

```
char st[] = "C Programming";
```

  - ' と " の使い分けに注意
  - 配列サイズを省略した場合はサイズは自動で決定される
  - 13文字に見えるが、最後にNULL文字（¥0）が追加されるため、stの配列長は14

- 文字列の配列はchar型の多次元配列となる

```
char tokai[][20] = {"Aichi", "Gifu", "Mie", "Shizuoka"};
```

  - 長さ20のchar配列を複数個（内容に依存、この例では4個）宣言
  - 長さ20文字に満たない分はNULLで埋められる

```
char j2e[][2][20] = {
  {"Ohayou gozaimasu", "Good morning"},
  {"Konnichiwa", "Hello"},
  {"Konbanwa", "Good evening"}};
```

  - 3次元以上の配列も作成できる

文字列の取り扱い

- C言語における文字列
  - 文字列のためのデータ型は存在しない
  - 文字（char）の配列で文字列を取り扱う
  - 文字列の終端として最後に必ずNULL文字（¥0 バックスラッシュゼロ）を入れる
- 具体例

```
char name[80];
name[0] = 'N';
name[1] = 'a';
name[2] = 'g';
name[3] = 'o';
name[4] = 'y';
name[5] = 'a';
name[6] = '¥0';
printf("%s¥n", name);
```

左の例では¥0を入れなくても恐らく期待通りの出力（Nagoya）となるが、未定義動作であり、正しい書き方ではない。
右の例で¥0の代入を怠るとname[6]のaが残ってしまい、2回目のprintfがNagoyaaになってしまう。

```
char name[80];
name[0] = 'F';
name[1] = 'u;
name[2] = 'k';
name[3] = 'u';
name[4] = 'o';
name[5] = 'k';
name[6] = 'a';
name[7] = '¥0';
printf("%s¥n", name);
name[0] = 'N';
name[1] = 'a';
name[2] = 'g';
name[3] = 'o';
name[4] = 'y';
name[5] = 'a';
name[6] = '¥0';
printf("%s¥n", name);
```

文字列操作に関する標準Cライブラリ

- 文字列操作ライブラリ
  - 使用するにはヘッダーファイルstring.hのインクルードが必要
- 文字列のコピー(copy string to another)

```
strcpy(ターゲット, ソース);
strncpy(ターゲット, ソース, バイト数);
```

  - NULLも含めてコピー
  - ソースの指定バイト数だけコピー
    - 指定バイト数にNULLがない場合はターゲットにNULLは挿入されない
- 文字列の連結(concatenate two strings)

```
strcat(ターゲット, ソース);
strncat(ターゲット, ソース, バイト数);
```

  - ターゲットのNULLを削除してその直後にソースを連結、最後にNULLを挿入（strncatではstr2をn文字だけ追加する）
- 文字列の比較(compare two strings)

```
strcmp(文字列1, 文字列2);
strncmp(文字列1, 文字列2, バイト数);
```

  - 文字列1が文字2に比べて小さいと0より小さい値、等しいと0、大きいと0より大きい値を返す
- 文字列の長さ(return length of string)

```
strlen(文字列);
```

  - 文字列の長さを返す（NULLはカウントしない）

strcpyやstrcatはターゲットが十分な大きさである必要があるため、strlenで確認することが重要となる

文字列入力にgets()を使ってはいけない

- 外部からの文字列入力について
  - 古い資料では（注意を促しながらも）外部からの文字列入力関数としてgets()を利用していることがある
  - しかし、gets()は範囲チェックをしないためプログラムが危険な動作をする可能性があり、C11規格にて廃止されている。
  - 現在の実用的なCプログラムでは、gets() を使う場面は存在しない。

- 代替関数
  - gets_s()

```
gets_s(文字型配列名, 文字数);
```

C11でもオプション扱いであり、使えるかどうかは環境依存

  - fgets()

```
fgets(文字型配列名, 文字数, 入力先);
```

fgetsで代替するのが一般的だが、改行コードまで読み込んでしまうなど挙動が異なる点があることに注意

キーボードからの文字列入力

- Enterキーが押されるまでキーボードから文字列を入力

```
#include <stdio.h>

int main(void)
{
  char str[80];

  printf("Input string (no more than 80-letter)¥n");
  fgets(str, 80, stdin);
  printf("%s¥n", str);

  return 0;
}
```

stdinは標準入力（キーボード）を表す

- 実行結果

```
$ ./a.out
Input string (no more than 80-letter)
programming
programming

```

キーボードで入力した様子

プログラムが出力した結果（最後にprintfが出力した追加の改行がある）

自動で決定された配列の大きさの確認

- strlen()で文字列の（終端NULL文字までの）大きさを取得できる
- sizeof演算子で配列の大きさを取得できる

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

int main(void){
  char st[80] = "C Programming";
  printf("st: sizeof= %zu, strlen= %zu¥n", sizeof(st), strlen(st));

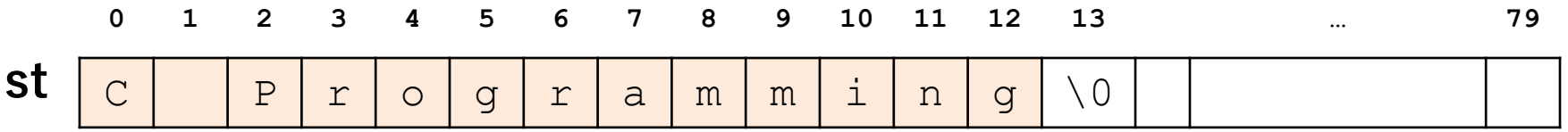
  char tokai[][20] = {"Aichi", "Gifu", "Mie", "Shizuoka"};
  printf("sizeof(tokai)= %zu¥n", sizeof(tokai));
  printf("sizeof(tokai[0])= %zu¥n", sizeof(tokai[0]));
  printf("sizeof(tokai[0][0])= %zu¥n", sizeof(tokai[0][0]));
  printf("Num of lines= %zu¥n", sizeof(tokai) / sizeof(tokai[0]));
  return 0;
}
```

strlen()を使うためにstring.hのインクルードが必要

```
$ ./a.out
st: sizeof= 80, strlen= 13
sizeof(tokai)= 80
sizeof(tokai[0])= 20
sizeof(tokai[0][0])= 1
Num of lines= 4
```

strlen関数とsizeof演算子の違い

```
char st[80] = "C Programming";
```



strlen(st) 13 （終端を表すNULL文字の前までの要素数=文字数）

sizeof(st) 80 （確保した配列自体の要素数）

↔ **sizeof(st[0])** 1 （st[0]単体のバイト数）

2次元配列とsizeof演算子

```
char tokai[][20] = {"Aichi", "Gifu", "Mie", "Shizuoka"};
```

このように宣言した場合、2次元配列の行数は自動で決定される、列数は指定が必要



tokai

	0	1	2	3	4	5	6	7	8	9	....	19
0	A	i	c	h	i	\0						
1	G	i	f	u	\0							
2	M	i	e	\0								
3	S	h	i	z	u	o	k	a	\0			

sizeof(tokai): 配列全体の長さ 80
sizeof(tokai[0]): 0行目の長さ 20
sizeof(tokai[0][0]): 0行目0要素目のバイト数 1

翻訳プログラム（文字列の比較）

- 次の仕様に基づいて日本語を英語に翻訳するプログラム
  - 文字列の配列で翻訳データベースを作成する
  - 日本語はローマ字（日本語のラテン文字表記）で取り扱う
  - キーボードから日本語の文字列を入力し、データベースに一致する単語が存在した場合はその英語訳を出力する

文字列の配列による翻訳データベース

```
char j2e[][2][20] = {
    {"Ohayou gozaimasu", "Good morning"},
    {"Konnichiwa", "Hello"},
    {"Konbanwa", "Good evening"}};
```

翻訳プログラムの例

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

char j2e[][2][20] = {{"Ohayou gozaimasu", "Good morning"},
    {"Konnichiwa", "Hello"}, {"Konbanwa", "Good evening"}};

int main(void)
{
  char buf[80];
  int i;
  printf("Please input Japanese Greetings: ");
  fgets(buf, sizeof(buf), stdin);
  buf[strlen(buf)-1] = '¥0';

  for (i = 0; i < sizeof(j2e)/sizeof(j2e[0]); i++) {
    if (strcmp(buf, j2e[i][0]) == 0) {
      printf("in English: %s¥n", j2e[i][1]);
      break;
    }
  }
  return 0;
}
```

strcmp()を使うためにstring.hのインクルードが必要

翻訳データベース（グローバル変数として定義）

- 入力受付

fgets()で改行コードまでキーボードから読み込み、最後(strlen()-1)の文字を改行コードからNULLに修正

- 文字列の検索

strcmp()でデータベースから一致するものを検索

実行結果例

```
$ ./a.out
Please input Japanese Greetings: Konbanwa
in English: Good evening
$ ./a.out
Please input Japanese Greetings: Ohayou gozaimasu
in English: Good morning
$ ./a.out
Please input Japanese Greetings: Oyasumi
```

文字列の比較

- 文字列の比較は自分でループ処理を書いて実装しても良いが、文字列比較用の関数が用意されている

```
char j2e[][2][20] = {
  {"Ohayou gozaimasu",    "Good morning"},
  {"Konnichiwa",          "Hello"},
  {"Konbanwa",            "Good evening"}
};

  for (i = 0; i < sizeof(j2e)/sizeof(j2e[0]); i++) {
    if (strcmp(buf, j2e[i][0]) == 0) {
      printf("in English: %s¥n", j2e[i][1]);
      break;
    }
  }
```

```
printf("%d¥n", sizeof(j2e));          → 120
printf("%d¥n", sizeof(j2e[0]));       → 40
printf("%d¥n", sizeof(j2e[0][0]));    → 20
```

```
sizeof(j2e)/sizeof(j2e[0]);           → 3
```

- 入力文字列bufがj2e[i][0](=日本語綴り)と一致したら、j2e[i][1](=英字綴り)を出力

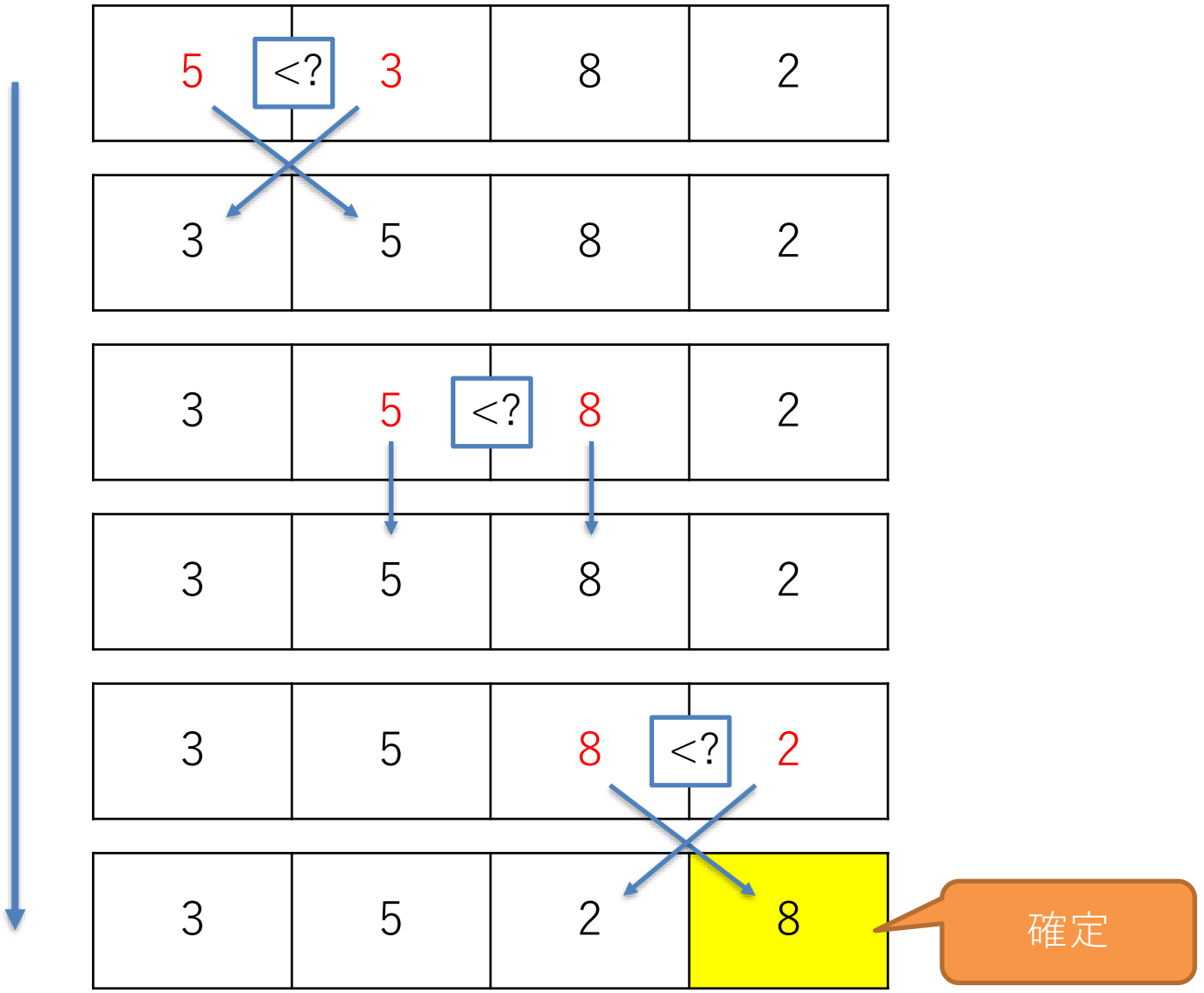
データ（配列）の並び替え（ソート）を考える

- 数値（整数）が格納された配列を小さい順または大きい順に並び替えることを考える
- ソートは基本的なアルゴリズムの1つであり、多数のアルゴリズムが提案・利用されている
  - バブルソート
  - 選択ソート
  - クイックソート
  - マージソート
  - カウンティングソート
  - 基数ソート
- ソートアルゴリズムごとに特徴（利点・欠点）がある
  - 平均的に高速、最悪ケースでもあまり遅くない、入力データの偏りに速度が影響を受けにくい、メモリ使用量が少ない、など

バブルソートのアルゴリズム

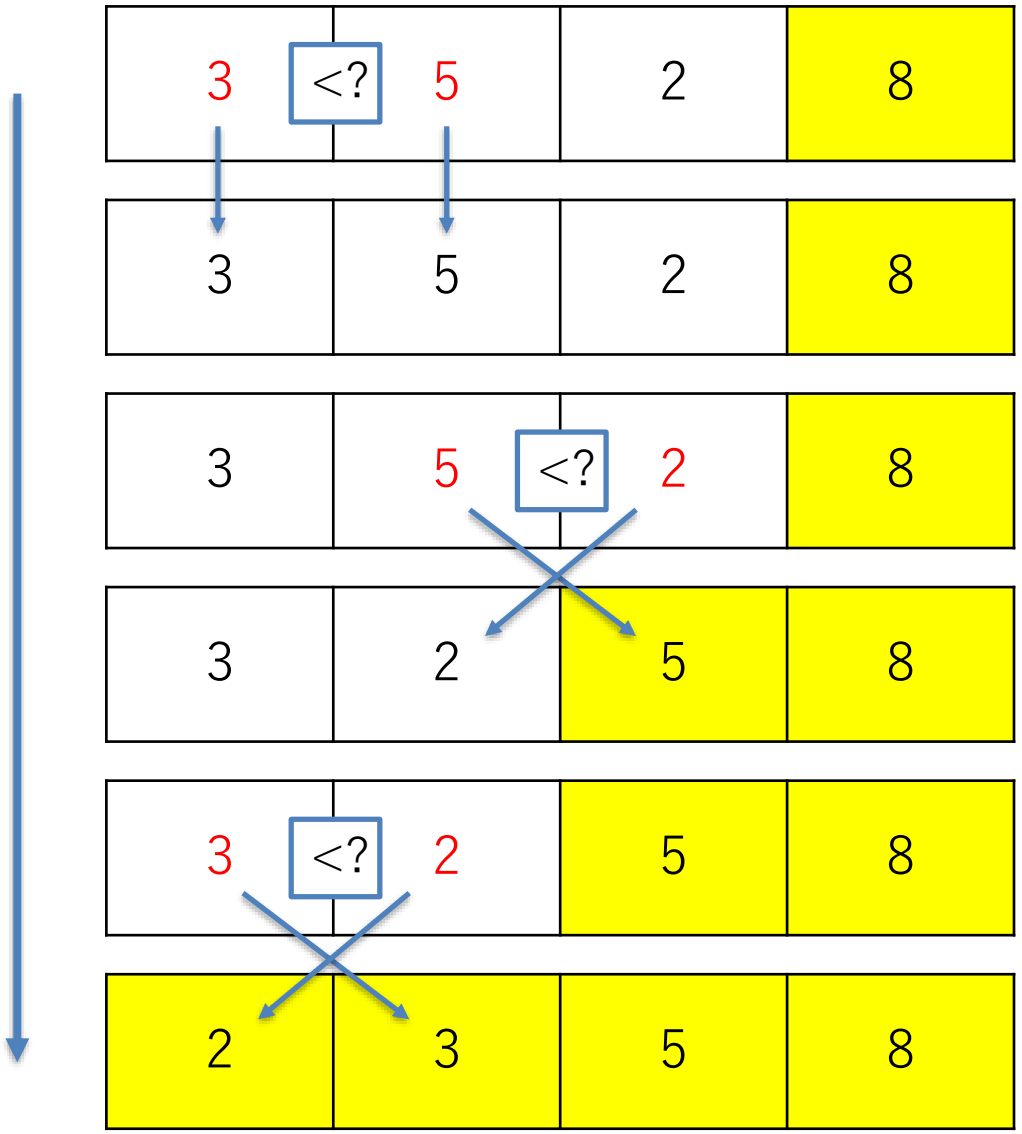
1. バブルソートで小さい順に並べる。配列aのi番目の要素をaiとするとき、隣り合う２つの要素を比べて、値を入れ替える
    1. a0とa1を比較、 a0が大きいならば値を入れ替え
    2. a1とa2を比較、 a1が大きいならば値を入れ替え
    3. …
    4. aN-2とaN-1を比較、 aN-2が大きいならば値を入れ替え
2. 範囲をa0からaN-2とし、1と同じ手順で並び替える
3. …
4. 範囲をa0からa1とし、同じ手順で並び替える

実例



実例

（つづき）

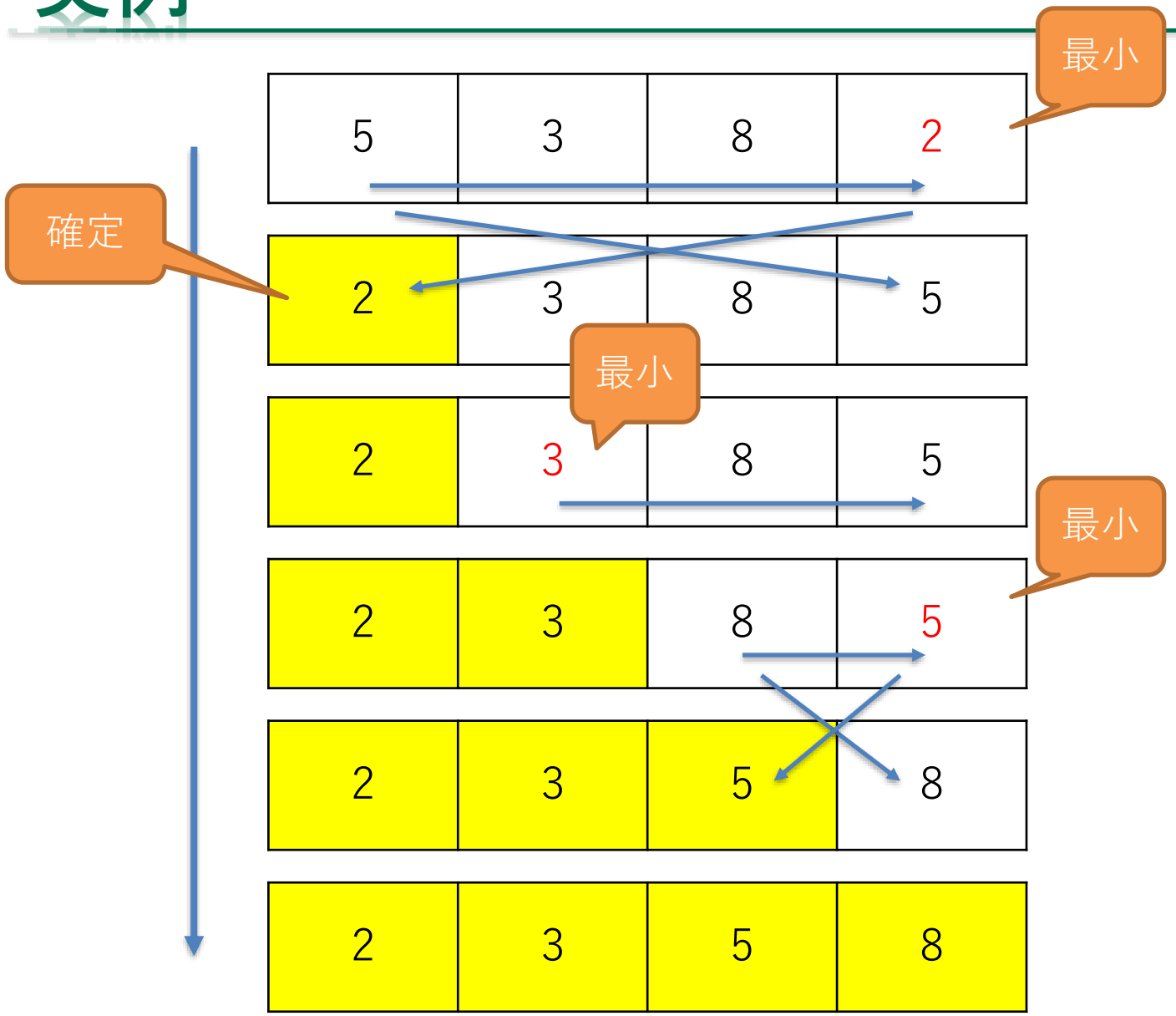


- もちろん、配列indexが大きな側から比較を行ったり、小さい値を優先的に確定したりしても良い

選択ソートによる並び替え

- アルゴリズム
  - 左端から右へ向かって全配列の値をチェックし、一番小さなものを左端に配置
  - 左端から2つ目の要素から同様に、全配列をチェックして一番小さなものを左端（確定したもののすぐ右）に配置
  - 右端まで繰り返せば終了
  - （もちろん大小反対や左右逆順でも良い）

実例



第３回 演習

EX-3.1

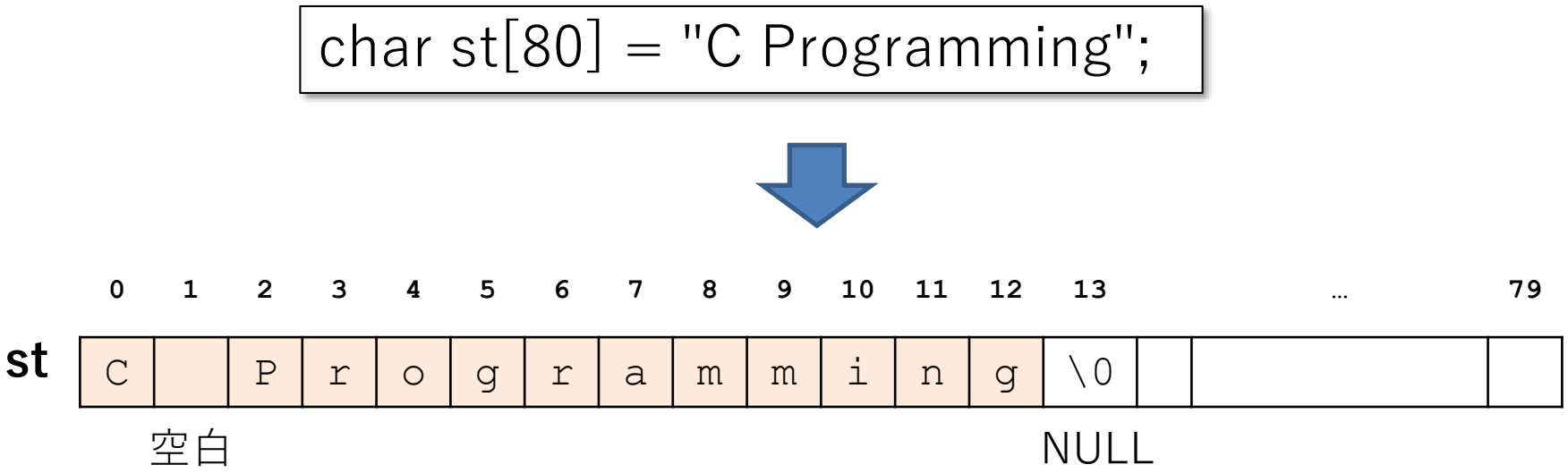
- 入力された文字列を区切り文字で単語に分割し、1, 4, 7, … 番目の単語を右揃え、2, 3, 5, 6, … 番目の単語を左揃えで出力する関数blingbangbangbornを作成せよ
  - 区切り文字は”-”（ハイフン）と”,”（カンマ）とする
  - 各単語は10文字以下としてよい（10より大きい時の例外処理もしなくていい）
  - 右揃えは、単語の長さと足して10になるように、左側にスペースを埋める
  - TACTに雛形を添付しています（ex-3-1.c）

正しく動作した例

```
$ ./bbbb
Input string (no more than 100-letter): Bling-Bang-Bang,Bling-Bang-Bang,Bling-Bang-Bang-Born
     Bling
Bang
Bang
     Bling          ⟸  スペース×5 + Bling
Bang
Bang
     Bling
Bang
Bang
      Born          ⟸  スペース×6 + Born
```

ヒント

- 文字列はcharの配列で、終わりに終端記号¥0（¥はバックスラッシュ）が入っている
  - 単語に分解するので、それぞれに終端記号が必要なはず
- （デバッグの際）printfのフォーマット指定子は、文字列なら%s、文字なら%c



現段階ではここまで考えなくていい

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

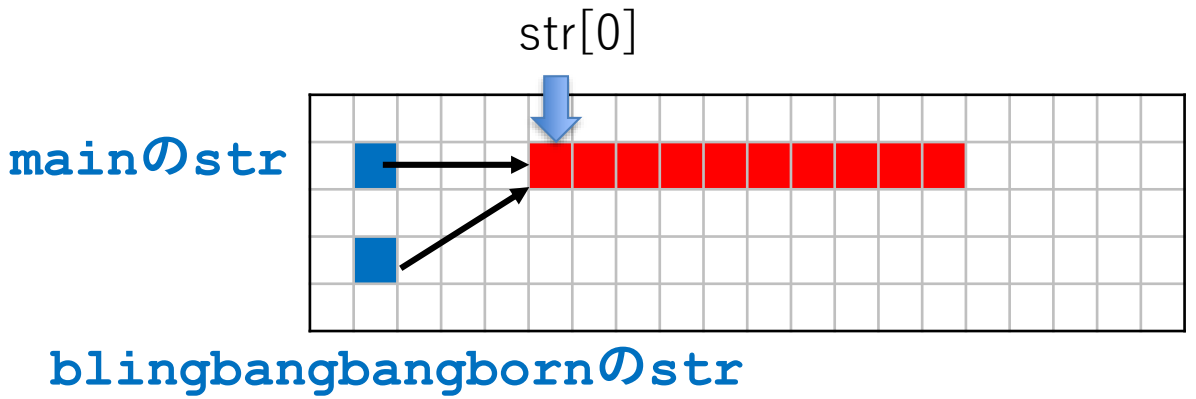
void blingbangbangborn(char* str);

int main(void)
{
  char str[101];
  printf("Input string (no more than 100-letter): ");
  fgets(str, sizeof(str), stdin);
  str[strlen(str)-1] = '\0';

  blingbangbangborn(str);

  return 0;
}

void blingbangbangborn(char* str)
{
  /* ここを実装 */
}
```



main関数でblingbangbangborn(str)が呼ばれたときに、mainのstrの値（つまりstr[0]のアドレス）がblingbangbangbornのstrにコピーされる。
blingbangbangbornの中では、あたかもstrをmain関数で宣言された配列のように扱える。

EX-3.2

- バブルソートと選択ソートを実装する
  - ソースが長いので、TACTに雛形を添付しています（ex-3-2.c）
  - TACTの雛形をベースに、bubble_sort 及び selection_sort 関数の実体を完成させる
- 仕様
  - コマンドライン引数から、ソートタイプとソートする数値の列を受け取る
  - ソートタイプはbまたはsで、b: バブルソート、s: 選択ソート とする
  - 数値の列は10個以下で、各数値は1以上9以下とする
    - 入力数値が1以上9以下であることを前提に実装して良い
  - ソートして小さい順に並べる
  - bubble_sort 及び selection_sort の関数の戻り値は、値の交換が起こった回数とする
  - 値の交換を行う度に、print_array関数を使って、現在の数値の状況を画面出力する
- 以下の実行時引数で結果を確認すること
  - b 9 8 7 6 5 4 3 2 1
  - s 9 8 7 6 5 4 3 2 1

出力例

```
$ ./a.out b 9 8 7 6 5 4 3 2 1
input array
*********
********
*******
******
*****
****
***
**
*


bubble sort start
*********
********
*******
******
*****
****
***
*
**


*********
********
*******
******
*****
****
*
***
**
```

```
（中略）

*
**
***
****
*****
******
*******
********
*********


sorted array
*
**
***
****
*****
******
*******
********
*********


#of sort steps = 36
```

ソートの途中経過はこの通りにならなくても良い

```
$ ./a.out s 9 8 7 6 5 4 3 2 1
input array
*********
********
*******
******
*****
****
***
**
*


selection sort start
*
********
*******
******
*****
****
***
**
*********


*
**
*******
******
*****
****
***
********
*********
```

```
（中略）

*
**
***
****
*****
******
*******
********
*********


sorted array
*
**
***
****
*****
******
*******
********
*********


#of sort steps = 4
```

ヒント

- バブルソート
  - 隣の値と比較＆交換する
  - 逆向きのfor文の書き方（必ずしも使わない）

```
for(i = n-1; i >= 0; i--)
```

- 選択ソート
  - 最小（最大）を見つけて値を交換する
  - 見つけた最小値の場所を覚えておけば、後で交換できる

提出

- 演習課題EX-3.1, EX-3.2
- TACTでソースファイルと実行結果（入力+画面に出力されたものをコピー＆ペーストしたテキストファイル）を提出
  - ソースファイルの名前は　学籍番号-3-{課題番号}.c　とすること
    - 例：252301111-3-1.c
  - 結果ファイルの名前は　学籍番号-3-{課題番号}.txt　とすること
    - 例：252301111-3-1.txt
  - 実行可能ファイル（gccにより生成されたファイル）の提出は不要
  - ２課題分計4ファイル提出
- 提出期限
  - **次々回（12/26）講義開始前まで**