プログラミング第一同演習

慶應義塾大学 理工学部 情報工学科
講義担当: 河野 健二
演習担当: 杉浦 裕太

再掲: 講義内容 (2/2) の変更

- 12/ 8: 配列とポインタ
  - 配列とポインタの関係
- 12/15: 文字列
  - 文字列，文字列とポインタの関係
- 12/22: 標準ライブラリ
  - 便利な関数の紹介
  - マクロや分割コンパイルなど

- ~~1/ 7~~, 12: プロジェクト課題
  - これまで習ったことを使って，
    実用っぽいプログラムを書いてみる
  - 実際には，画像処理をやってみる

1/7（土）は休講.

  - プロジェクト課題は 1/12 のみ！

- 1/19: 最終回
  - 少し上級向けの話
  - 言語処理系に関するいろいろ

最終回は課題は出しません
でも，演習時間には TA がきます
終わっていない課題やったり，
質問したり，うまく活用してください

本日の内容

sslab system software

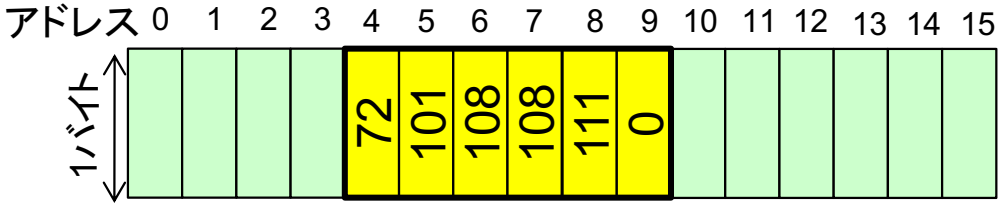
- C 言語における文字列
  - 文字列とポインタの関係
  - 文字列を扱うための標準ライブラリの紹介
- 静的なローカル変数
  - （発展的な話題として）extent の考え方を紹介

文字列

- 文字が並んだものを文字列という
- C 言語では“と”で囲んだ文字の並びのこと
  - `printf("Hello, World");` の `"Hello, World"` は文字列
  - `scanf("%d");` の `"%d"` は文字列
  - `"a"` は文字列
  - `""` は長さ 0 の文字列．ヌル (null) 文字列という
- `'` と `'` で囲んだ 1 文字とは別物
  - `'a'` と `"a"` は別物
  - 詳細は後で

文字列とメモリの関係

- 文字列 "Hello" はメモリ上では次のようになる



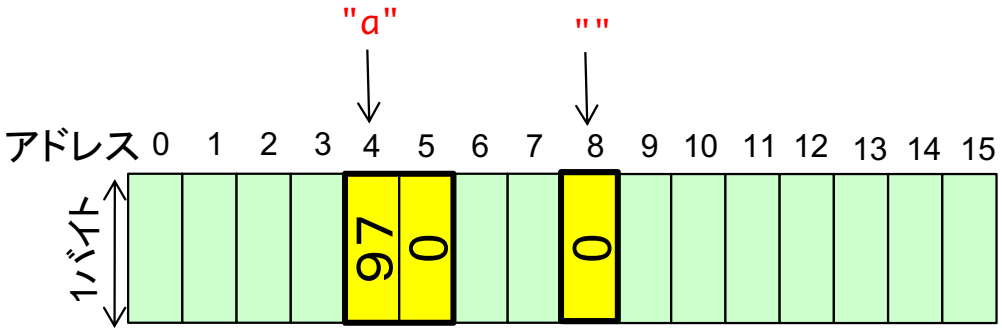
- ひとつひとつの文字は ASCII コードになる
  - H → 72, e → 101, l → 108, o → 111
- 文字列の終端を示すため，文字列の最後に 0 を置く
  - この 0 は文字の '0' ではなく数字の 0
  - 文字列の終端を示す 0 をヌル文字という
    - 長さ 0 のヌル文字列とは別物

文字列は可変長（文字数が最初から決まっているわけでない）ため

文字とメモリの関係: 具体例

- 文字列 "a"
  - 文字 'a' に対応する ASCII コードは 97
  - 文字列の終端を表すヌル文字を最後におく

- 長さ 0 の文字列 "". 空の文字列ともいう
  - いきなり文字列の終端を表すヌル文字をおく



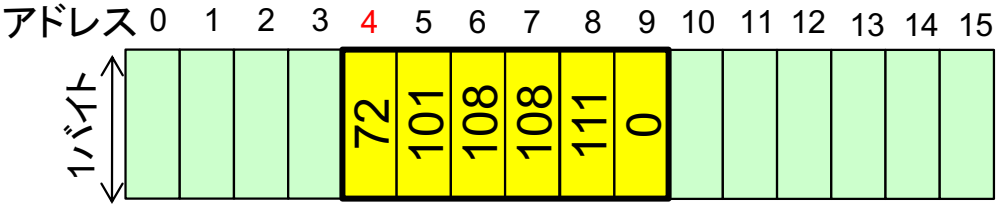
ASCIIコード表

	0	16	32	48	64	80	96	112
+0	NULL	DLE	SP	0	@	P	`	p
+1	SOH	DC1	!	1	A	Q	a	q
+2	STX	DC2	“	2	B	R	b	r
+3	ETX	DC3	#	3	C	S	c	s
+4	EOT	DC4	$	4	D	T	d	t
+5	ENQ	NAK	%	5	E	U	e	u
+6	ACK	SYN	&	6	F	V	f	v
+7	BEL	ETB	‘	7	G	W	g	w
+8	BS	CAN	(	8	H	X	h	x
+9	HT	EM	)	9	I	Y	i	y
+10	NL	SUB	*	:	J	Z	j	z
+11	VT	ESC	+	;	K	[	k	{
+12	NP	FS	,	<	L	¥	l	|
+13	CR	GS	-	=	M	]	m	}
+14	SO	RS	.	>	N	^	n	~
+15	SI	US	/	?	O	_	o	DEL

赤字: 制御文字
青地: スペース
黒字: 通常文字

文字列の型

- 文字列はポインタを使って参照する
  - ポインタということはアドレスを使って参照するということ
  - 文字列の先頭アドレスを使う
- 例:
  - "Hello" という文字列の先頭アドレスは 4



72 → H
101 → e
108 → l
111 → o
0 → ヌル文字

  - アドレス 4 番地に入っているのは char 型の値
    - ひとつひとつの文字は char 型となっている
  - したがって，"Hello" の型は char * となる
    - char 型を指すポインタということになる

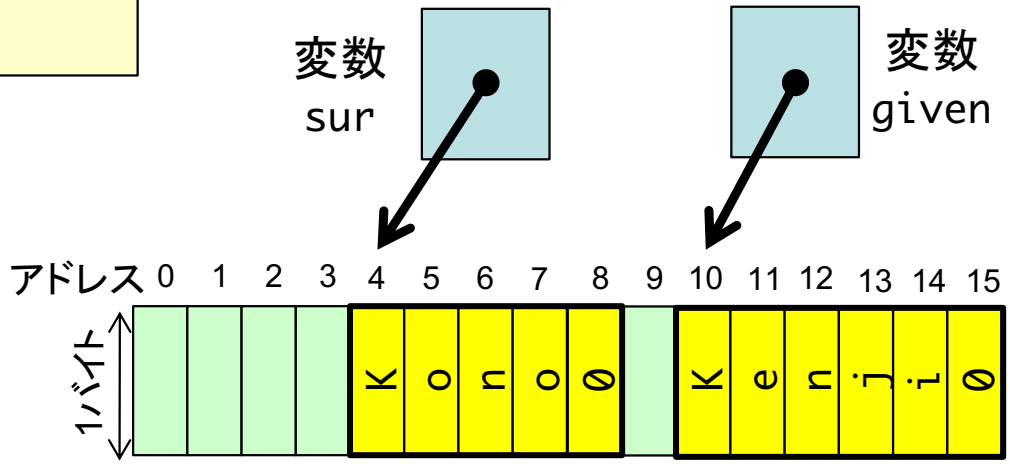
文字列の宣言(1/2)

- 文字列 "Kono", "Kenji" を参照する変数の宣言

```
int main()
{
    char *sur = "Kono";
    char *given = "Kenji";

    ...
}
```

変数 sur には文字列 "Kono" の先頭アドレス,
変数 given には文字列 "Kenji" の先頭アドレスが入る



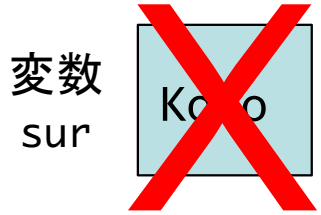
文字列の宣言(2/2)

- 文字列 "Kono", "Kenji" を参照する変数の宣言

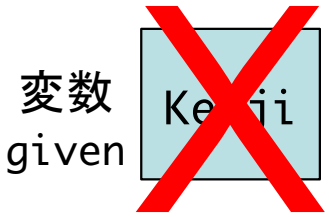
```
int main()
{
    char *sur = "Kono";
    char *given = "Kenji";

    ...
}
```

変数 sur には文字列 "Kono" の先頭アドレス,
変数 given には文字列 "Kenji" の先頭アドレスが入る



絶対にこうなっている
と思ってはいけない！



C 言語には“文字列型”のようなものは存在しない！

文字列の表示

- printf() では "%s" を使って表示する
  - 実引数には文字列の先頭アドレスを渡す

```
int main()
{
    char *sur = "Kono";
    char *given = "Kenji";

    printf("My name is %s %s¥n", given, sur);

    return 0;
}
```

  - printf("My name is %s %s¥n", *given, *sur) は間違い！
    - *given はポインタ given が指すアドレスにあるひと文字になる

ヌル文字で終わるのが文字列

- 下のプログラムを実行すると・・・

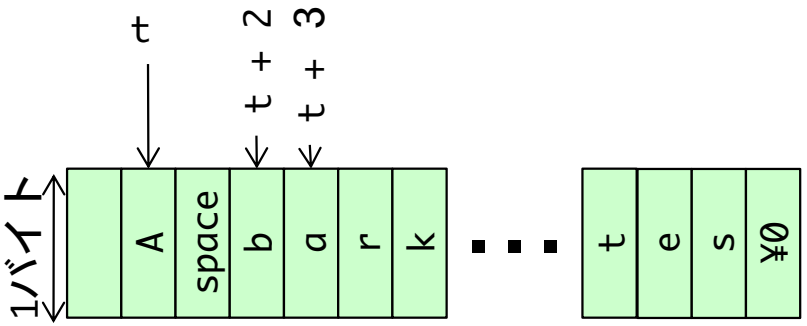
```
A barking dog never bites
barking dog never bites
arking dog never bites
```

と表示される．理由を考えてみよう

```
int main()
{
    char *s = "A barking dog never bites";
    char *t = s;

    printf("%s¥n", t);
    printf("%s¥n", t + 2);
    printf("%s¥n", t + 3);

    return 0;
}
```



ポインタの指す先から，ヌル文字（¥0）までが文字列と解釈される

'¥0' の ASCII コードは 0. すなわちヌル文字

例題: 大文字・小文字変換

- 標準入力から文字列を読み込み，アルファベットの大文字は小文字に，小文字は大文字に変換しなさい．アルファベット以外は変換しなくてよい

- 実行例:

```
$ cat data.txt
All work and no play makes Jack a dull boy.

$ ./reidai < data.txt
All work and no play makes Jack a dull boy.
aLL WORK AND NO PLAY MAKES jACK A DULL BOY.
```

入力となる文字列を表示

大文字小文字変換の結果を表示

```c
#include <stdio.h>
#define MAX_LINE      256

void conv(char *s)
{
    while (*s != '¥0') {
        if ('a' <= *s && *s <= 'z') {
           *s = *s - 'a' + 'A';
        } else if ('A' <= *s && *s <= 'Z') {
           *s = *s - 'A' + 'a';
        }
        s++;
    }
}
```

最初，s は入力された文字列の先頭を指す

文字列の終端まで繰り返すということ

ポインタ s の値を一つ増やす．
つまり，次の文字を調べる

文字列を入れるための配列を宣言

```c
int main()
{
    char line[MAX_LINE];

    fgets(line, MAX_LINE, stdin);
    printf("%s", line);
    conv(line);
    printf("%s", line);

    return 0;
}
```

fgets() を使うと標準入力から
文字列を読み込むことができる
（詳細は後述）

文字列の入力: fgets()

- `char *fgets(char *buffer, int size, FILE* fp)`
  - 引数:
    - `buffer`: 読み込んだ文字列を入れる領域の先頭アドレス
    - `size`: 読み込める最大の文字数
    - `fp`: 入力元を指定する. 今回は標準入力を表す `stdin` と書いておけばよい. 以下の説明は標準入力から読み込むとして説明
  - 動作:
    - 標準入力から改行文字 (`'¥n'`) までを `buffer` に読み込む
    - 改行文字 (`'¥n'`) も読み込む
    - 文字列の最後には `'¥0'`（ヌル文字）を入れる
  - 戻り値の意味:
    - 読み込む行がなくなると NULL を返す

fgets() の使い方

- 文字列を読み込むための配列を宣言する
  - char line[MAX_LINE];
- 読み込む最大の文字数を指定して読み込む
  - fgets(line, MAX_LINE, stdin);
  - fgets の最初の引数の型は char * 型．そこに char 型の配列を渡している
  - 配列の要素数は MAX_LINE 以上，確保されていること

文字列を入れるための配列を宣言

```
int main()
{
    char line[MAX_LINE];

    fgets(line, MAX_LINE, stdin);
    printf("%s", line);
    conv(line);
    printf("%s", line);

    return 0;
}
```

fgets() を使うと標準入力から文字列を読み込むことができる

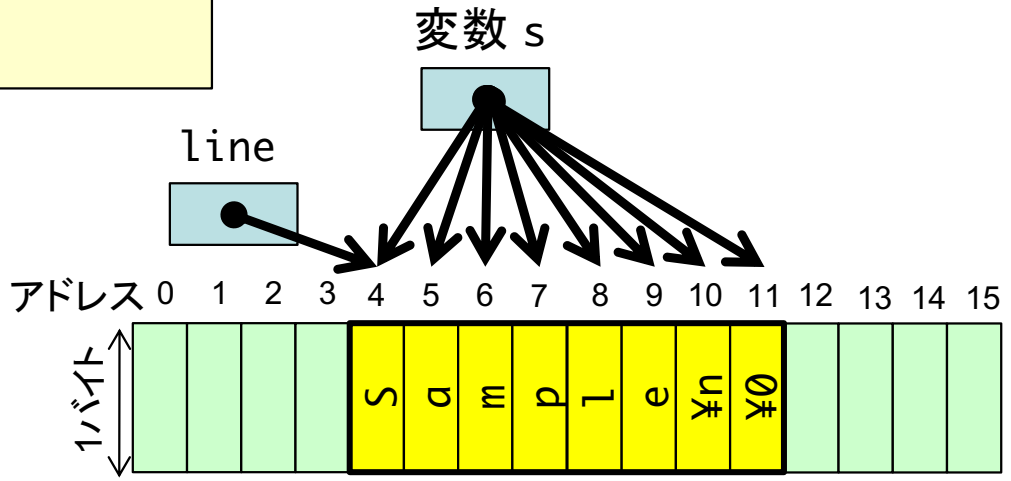
```
#include <stdio.h>
#define MAX_LINE      256

void conv(char *s)
{
    while (*s != '¥0') {
        if ('a' <= *s && *s <= 'z') {
           *s = *s - 'a' + 'A';
        } else if ('A' <= *s && *s <= 'Z') {
           *s = *s - 'A' + 'a';
        }
        s++;
    }
}
```

最初，s は入力された文字列の先頭を指す

文字列の終端まで繰り返すということ

ポインタ s の値を一つ増やす．
つまり，次の文字を調べる



strlen(): 文字列の長さを返す

- ライブラリ関数 strlen(): 文字列の長さを返す

  int strlen(char *s)

  - s : 文字列へのポインタ

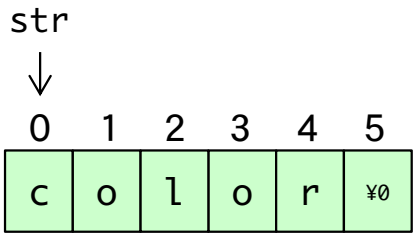
```
char *str = "color";
int len;

len = strlen(str);
printf("len = %d¥n", len);
```

文字列 color の長さは 5

strlen() を使うと文字列の長さが得られる

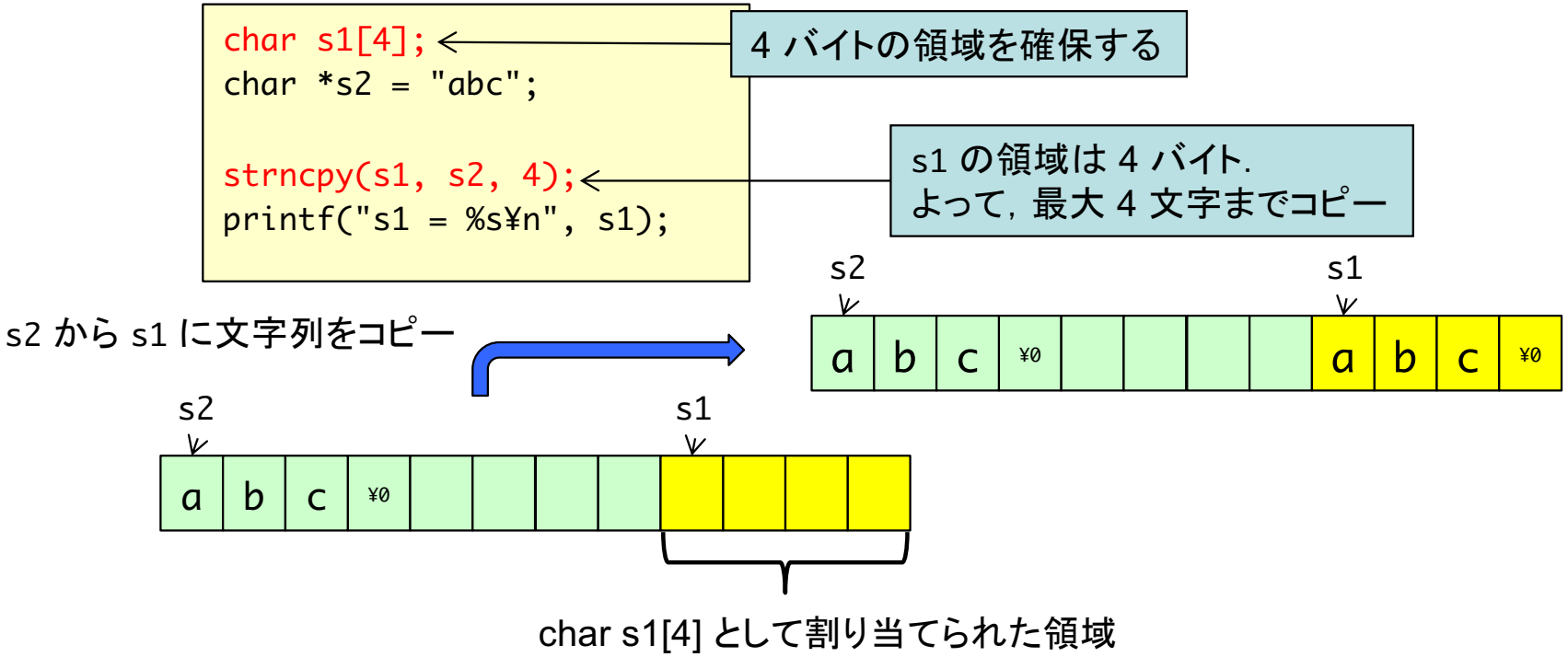
len = 5 と表示される



str が指す文字列の長さ = 5

strncpy(): 文字列のコピー

- char *strncpy(char *s1, char *s2, size_t n)
  - 文字列 s2 をポインタ s1 の指す領域にコピーする
  - ただし，最大 n 文字までしかコピーしない



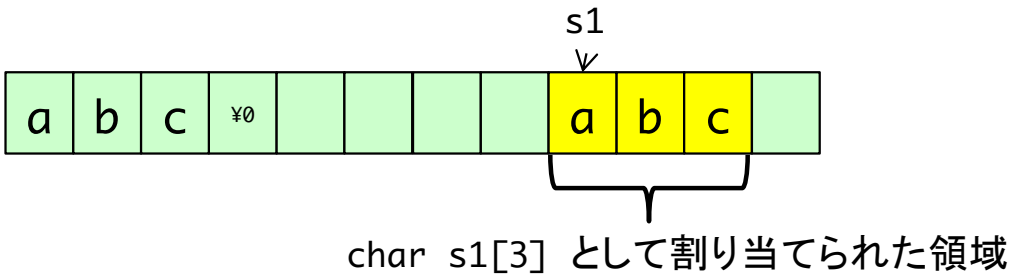
strncpy() の注意点

- strncpy(s1, s2, n) を実行したとき:
  - ポインタ s1 の先は n バイト以上の領域が割り当てられていなければならない

```
char s1[3];
strncpy(s1, "abcd", 4);      // 間違い．実行時におかしくなる
```

  - ヌル文字がコピーされないことがある

```
char s1[3];
strncpy(s1, "abc", 3);       // s1 にヌル文字がコピーされない
```



s1 の指す先は，文字列の終端を示す '¥0' がない！

strncpy() の正しい使い方

sslab system software

- ヌル文字のための 1 バイトを空けておき，
  自分でヌル文字を書き込むようにしておく
  - ```
    char s1[N];
    ...
    strncpy(s1, s2, N - 1);  // ヌル文字のために 1 バイト空けておく
    s1[N-1] = '¥0';     // 自分でヌル文字を追加する
    ```
- 慣れないうちは十分に大きな領域をとっておくのが安全！
    ```

文字列の比較(1/2)

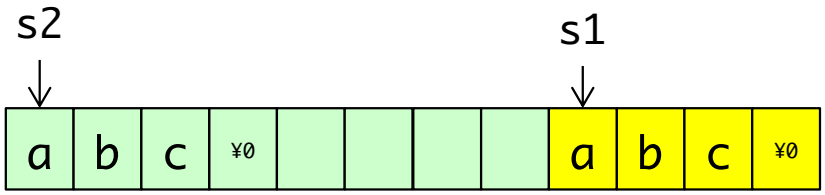
- 文字列が等しいかどうかを比較する
  - 次のプログラムでは文字列同士の比較にはならない
  - 文字列の先頭アドレスが等しいかどうか比較しているだけ

```
char s1[10], *s2 = "abc";
strncpy(s1, s2, 10);

if (s1 == s2) {
     printf("good¥n");
} else {
     printf("bad¥n");
}
```

s1 に s2 をコピー

アドレス s1 とアドレス s2 の比較.
よって, これは偽.



文字列の比較(2/2)

sslab system software

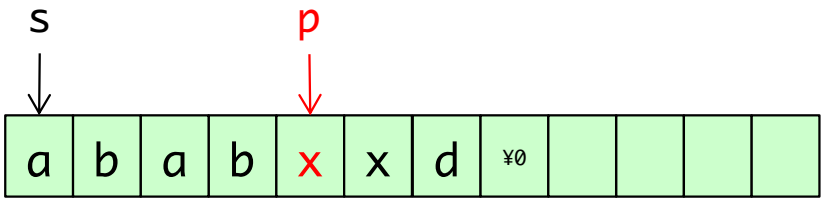
- `int strcmp(char *s1, char *s2)`
  - 文字列 s1 と s2 が等しければ 0 を返す
    - `s1 == s2` ではなく，`strcmp(s1, s2) == 0` とすればよい
    - `s1` の指す文字列と，`s2` の指す文字列とを比較する

strchr(), strrchr(): 文字を探す

- char *strchr(char *s, int c)
  - 文字列 s の中に最初に出現する文字 c (ASCII コードが c となる文字) の位置を返す
  - 文字 c が見つからなかったら NULL を返す

```
char *p, *s = "ababxxd";

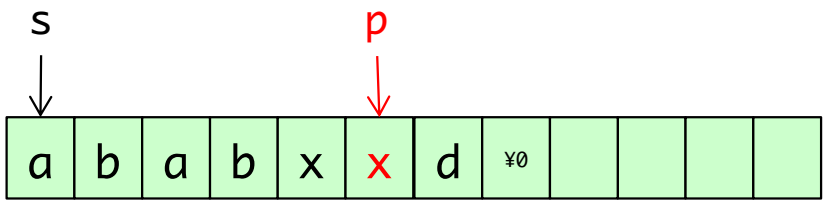
p = strchr(s, 'x');
```



- char *strrchr(char *s, int c)
  - 文字列 s 中の最後に出現する文字 c (ASCII コードが c となる文字) の位置を返す
  - 文字 c が見つからなかったら NULL を返す

```
char *p, *s = "ababxxd";

p = strrchr(s, 'x');
```



strtol(): 文字列表記を数値に変換

- 一般形

```
long strtol(char *str, char **endptr, int base)
```

  - str: 文字列へのポインタ
  - endptr: NULLと書いておけばよい
  - base: 何進数かを指定する
- 10進数文字列を数値に変換するには以下のようにする

```
char str[] = "123";
int i;

i = strtol(str, NULL, 10);
```

例題: 最長行と最短行

- 標準入力から何行かにわたるテキストが与えられるものとし，その中で最も長い行と最も短い行を，その長さと共に表示すること

```
% cat data.txt
Excuse me.
Yes. Can I help you?
Where is the TOKYO station?
Oh, I'm sorry. I'm stranger here.
No. Thank you just the same.
%
% ./sample < data.txt
 longest line ( 34): Oh, I'm sorry. I'm stranger here.
shortest line ( 11): Excuse me.
%
```

例題: プログラム (1)

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

#define    MAXLINE         256      /* 行の長さの最大 */

int mygetline(char [], int);

int main()
{
    int len, maxlen = 0, minlen = MAXLINE;
    char line[MAXLINE], maxstr[MAXLINE], minstr[MAXLINE];
```

- len は現在読み込んだ行の長さ，line[ ] は読み込んだ文字列用
- maxlen は最も長い行の長さ，minlenは最も短い行の長さ
- maxstr[ ] は最も長い行の文字列用，minstr[ ] は最も短い行の文字列用

例題: プログラム (2)

```
// continued from previous page

while ((len = mygetline(line, MAXLINE)) > 0) {


}
```

- mygetline() で line[ ] に1行のデータを読み込み，行の長さが lenに代入される
- len が正の間，ループする

例題: プログラム (3)

```
    // continued from previous page
    while ((len = mygetline(line, MAXLINE)) > 0) {
        if (len > maxlen) {
            strncpy(maxstr, line, MAXLINE);
            maxlen = len;
        }

        if (len < minlen) {
            strncpy(minstr, line, MAXLINE);
            minlen = len;
        }
    }
    if (maxlen != 0) {
        printf("longest line (%3d): %s", maxlen, maxstr);
        printf("shortest line (%3d): %s", minlen, minstr);
    }
}
```

・maxlen は今までの最長行の長さを保持.
・より長い行が入力されたら，copy()でその行を maxstr[ ] にコピーし，maxlen を更新する.

・最短行についても，最長行の場合と同様の処理

・結果の表示
(特に説明すべきことはない)

例題: プログラム (4)

```
int mygetline(char buf[], int max)
{
    char *p;

    if (fgets(buf, max, stdin) != NULL) {
        p = strrchr(buf, '¥n');
        if (p != NULL) {
            *p = '¥0';
        }
        return strlen(buf);
    } else {
        return 0;
    }
}
```

fgets() を用いて改行文字までを読み込む

改行文字を取り除く

読み込んだ文字列の長さを返す

fgets() で読み込めなかったら 0 を返す

ローカル変数の復習

- ローカル変数は・・・
  - 関数の実行を開始したときに，変数が作られ，
  - 関数の実行が終了したときに，変数が捨てられる
    - C の用語では自動変数 (auto 変数) という

```
void foo()
{
    int x = 3;
    printf( “x = %d¥n” , x);
    x++;
}

int main()
{
    foo();
    foo();
    …
}
```

関数の実行開始時に x は作られる

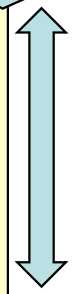
関数の実行が終わると x は捨てられる

2 回目の呼び出しでも
x = 3
が表示される

発展的な内容：変数のエクステント

- 変数が存在する期間のことを指す
  - スコープとは異なる概念
- 通常のローカル変数は・・・
  - 関数の実行を開始したときに，変数が作られ，
  - 関数の実行が終了したときに，変数が捨てられる
    - 「通常のローカル変数のエクステント (extent) は，関数の開始から終了まで」

ここを実行中も
変数 x は
（アクセスできないが）
存在する

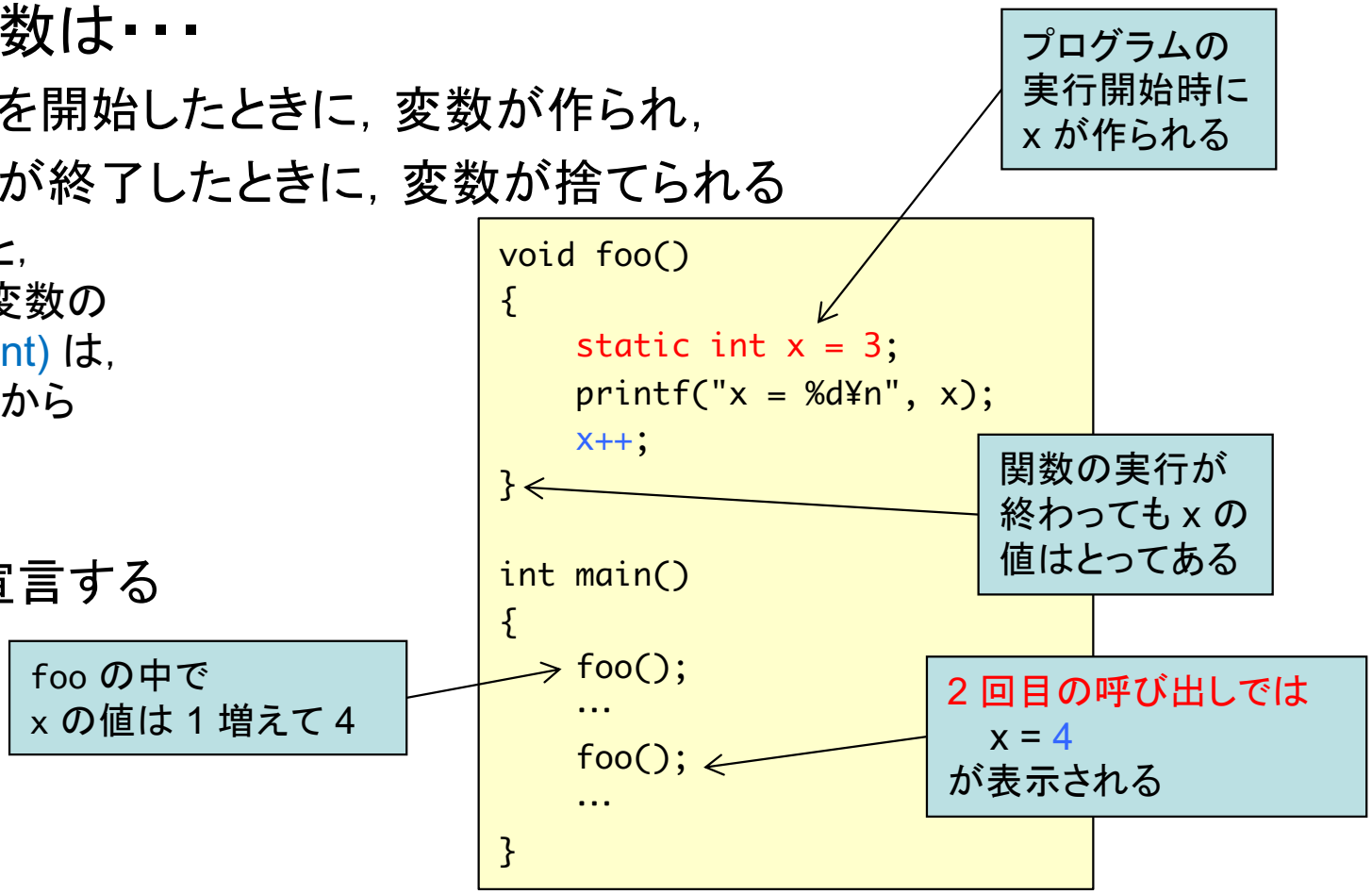


```
void bar()
{
      for (…) {
            時間のかかる処理
      }
}

void foo()
{
     int x = 3;
     bar();
     printf("x = %d¥n", x);
}
```

静的なローカル変数

- 静的なローカル変数は・・・
  - プログラムの実行を開始したときに，変数が作られ，
  - プログラムの実行が終了したときに，変数が捨てられる
    - かっこつけて言うと，
      「静的なローカル変数の
      エクステント (extent) は，
      プログラムの開始から
      終了まで」
  - 変数宣言の前に
    static と書いて宣言する



グローバル変数との違いは？

- 次の二つのプログラムは同じ結果を返す
  - x を静的なローカル変数からグローバル変数に移す
  - x のスコープが違うことに注意

```
void foo()
{
    static int x = 3;
    printf("x = %d¥n", x);
    x++;
}

int main()
{
    foo();
    …
    foo();
    …
}
```

変数 x の
スコープ

```
int x = 3;
void foo()
{
    printf("x = %d¥n", x);
    x++;
}

int main()
{
    foo();
    …
    foo();
    …
}
```

変数 x の
スコープ

静的なローカル変数の例

- 関数が呼び出された回数をカウントする

```
void foo()
{
    static int count = 0;
    printf("foo is called %d times¥n", ++count);
}

int main()
{
    foo();
    …
    foo();
    …
    foo();
    …
    return 0;
}
```

foo is called 1 times と表示

foo is called 2 times と表示

foo is called 3 times と表示

補足:
静的な変数は自動的に 0 に初期化される.
ただし, その場合でも自分で初期化するのがよい

Quiz

- 実行結果はどうなるでしょう？

```
w = 101, x = 401, y = 201, z = 301
w = 102, x = 401, y = 202, z = 301
x = 400
```

```
int w = 100;
void f(int x)
{
    static int y = 200;
    int z = 300;
    w++; x++; y++; z++;
    printf("w = %d, x = %d, y = %d, z = %d¥n", w, x, y, z);
}
int main()
{
    int x = 400
    f(x);
    f(x);
    printf("x = %d¥n", x);
    return 0;
}
```

まとめ

- C 言語における文字列
  - メモリ上に並んだ ASCII コードの列．ただし，ヌル文字で終わる
  - 文字列は `char *` 型のポインタで指し示す
  - 文字列を値として格納するような “文字列型” は存在しない
- 標準ライブラリの紹介
  - `fgets`, `strncpy`, `strcmp` など
- 静的なローカル変数
  - 発展的な話題として extent の考え方も紹介