プログラミング基礎

http://bit.ly/prog2d

文字列とポインタ

後期 第5週

2017/10/23

文字列の構造（p.317）

文字列は文字（char型）の配列構造で表現されます。

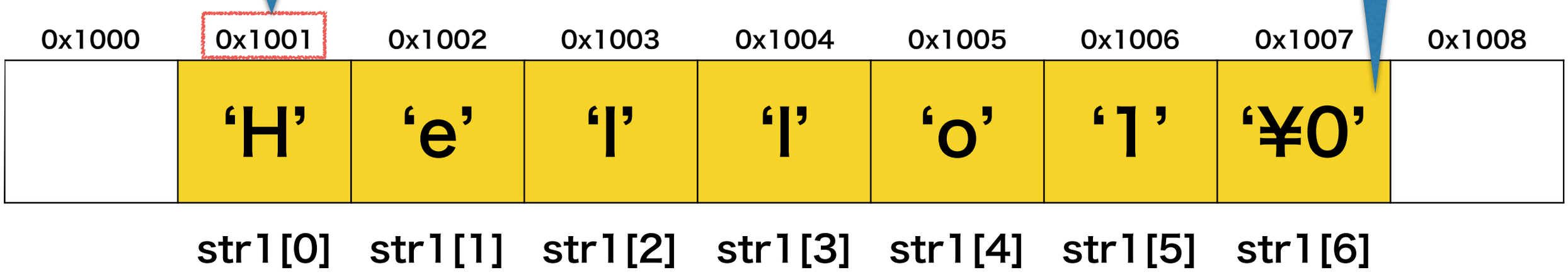
【例1】

```
char str1[] = "Hello1";
printf("str1: %s\n", str1);
```

char型の配列str1が宣言され、
初期値として文字列 “Hello1”が代入される

char型は1バイトなので、配列のアドレスも
1バイトずつ増えていく（p.318 図10-7上）

文字列は終端文字
（NULL文字）で終わる



文字列をポインタで参照する

文字列は、char型のポインタを使って参照することができます。

char型のポインタstr2が宣言され、
文字列 “Hello2”の先頭要素を参照する

【例2】

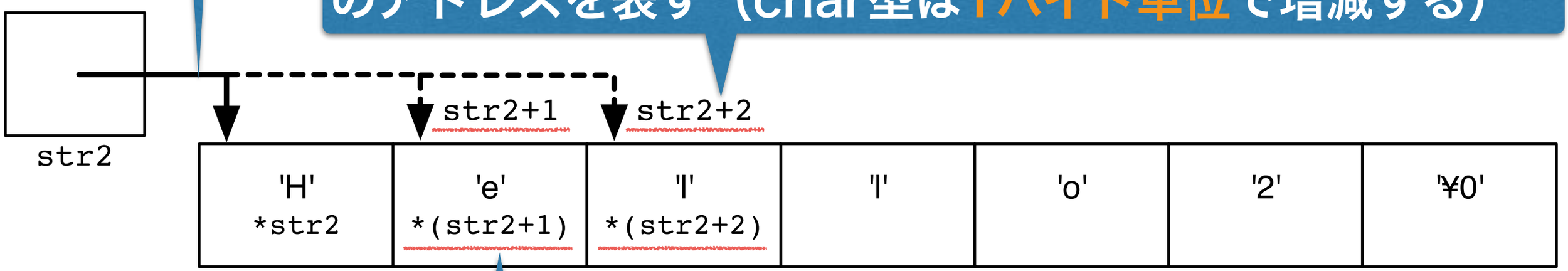
```
char *str2 = "Hello2";
printf("str2: %s\n", str2);
```

ポインタstr2がchar型の配列の先頭要素を参照する
→ ポインタstr2が文字列の先頭文字を参照する

文字列への参照のイメージ

配列の先頭要素、つまり先頭文字を参照している (p.318 図10-7下)

アドレスを加算/減算すると、1つ次の要素/1つ前の要素のアドレスを表す（char型は1バイト単位で増減する）



加算/減算したアドレスに対して、間接参照演算子「*」を使うと、そのアドレスに格納されている文字（つまりchar型1文字）が得られる

配列とポインタの違い①（p.319）

文字列を配列で扱う場合、配列宣言と同時に初期値となる文字列を代入できますが、後からその配列に、改めて文字列全体を代入し直すことはできません。

配列str3に、あらためて文字列を代入しようとすると、コンパイルエラーになる (p.204)

【例3】

```
char str3[] = "Hello3";
str3 = "World3";
str3[3] = 'A';
```

配列の要素1つずつ、つまり1文字ずつなら代入できる

配列とポインタの違い①（p.319）

同様の事を、int型の配列で考えてみると理解しやすい

配列を初期化する (p.190)

配列を後からまとめて代入できない
（エラーになる）

【例4】

```
int x[] = {5, 3, 1};
x = {6, 4, 2};
x[2] = 10;
```

配列の要素1つずつなら代入できる

配列とポインタの違い①（p.319）

文字列をポインタで扱う場合、後から文字列を代入するということは「別の文字列へ参照を変更する」ということになり、文字列全体の代入が可能となります。

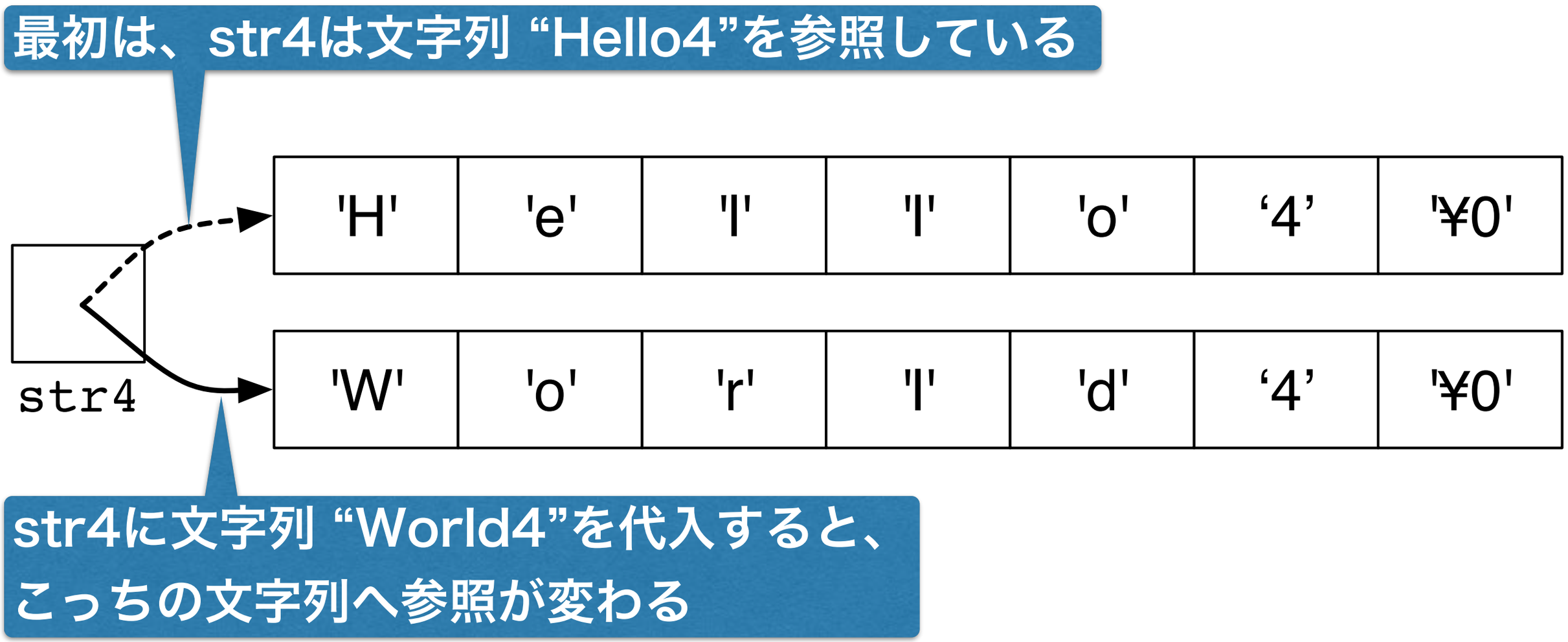
ポインタstr4が、文字列 “Hello4” の先頭文字を参照する

【例5】

```
char *str4 = "Hello4";
str4 = “World4";
```

str4が、文字列 “World4” の先頭文字を参照するように変更された

ポインタによる文字列代入のイメージ



【補足】どこからも参照されなくなった “Hello4”は、
明示的に削除（解放）されない限り、プログラム実行中
はメモリ上に残ってしまう（C言語の場合）

配列とポインタの違い②（p.321）

文字列をscanf()などで入力する場合、文字列を格納する領域を事前に配列で宣言しておく必要があります。

配列str5に、入力された文字列が格納される

【例6】

```
char str5[100];
char *str6;
printf("str5 > "); scanf("%s", str5);
printf("str5: %s\n", str5);
printf("str6 > "); scanf("%s", str6);
```

ポインタstr6はどこも参照していないため、入力された文字列を格納する場所がなく、実行時にセグメントエラーとなる。（コンパイル時にエラーにはならない）

文字列の配列（p.322）

複数の文字列をまとめて配列として扱えます。

char型の2次元配列（3×20）を宣言する
つまり、20文字（NULL文字含む）の文字列を3つ格納できる

3つの文字列を初期値で代入する

【例7】

```
char str7[3][20] = {"Hello", "Goodbye", "Thankyou"};
int i;
for(i=0; i<3; i++) {
    printf("str7[%d]: %s\n", i, str7[i]);
}
```

str7[0]（0番目の先頭文字）から
str7[2]（2番目の先頭文字）まで繰り返し出力する

文字列の配列のイメージ（p.323）

str7[0], str7[1], str7[2]が、それぞれ20個の要素の配列（20文字分の文字列）を持っている

'H'	'e'	'l'	'l'	'o'	'¥0'	-----													
str7[0][0]	str7[0][1]	str7[0][2]	str7[0][3]	str7[0][4]	str7[0][5]	str7[0][6]													str7[0][19]
'G'	'o'	'o'	'd'	'b'	'y'	'e'	'¥0'	-----											
str7[1][0]	str7[1][1]	str7[1][2]	str7[1][3]	str7[1][4]	str7[1][5]	str7[1][6]	str7[1][7]	str7[1][8]											str7[1][19]
'T'	'h'	'a'	'n'	'k'	'y'	'o'	'u'	'¥0'	-----										
str7[2][0]	str7[2][1]	str7[2][2]	str7[2][3]	str7[2][4]	str7[2][5]	str7[2][6]	str7[2][7]	str7[2][8]	str7[2][9]										str7[2][19]

ポインタによる文字列の配列（p.323）

文字列をポインタで扱うと、ポインタの配列としても書くことができます。

char型のポインタを3つ持つ配列を宣言する
つまり、文字列を3つ参照できる

3つの文字列を初期値で参照する

【例8】

```
char *str8[3] = {"Hello", "Goodbye", "Thankyou"};
int i;
for(i=0; i<3; i++) {
    printf("*(str8+%d): %s\n", i, *(str8+i));
}
```

str8のi番目のポインタが参照している文字列を出力する
「str8[i]」という書き方も可能 (p.324)

ポインタによる配列のイメージ（p.325）

文字列の操作（p.326）

文字列を扱う標準ライブラリ関数を使うと、文字列の色々な操作を活用することができるようになります。

主な標準ライブラリ関数（p.326 表10-2）

- strlen()　・・・文字列の長さを調べる
- strcpy()　・・・文字列をコピーする
- strcat()　・・・文字列を結合する
- strcmp()　・・・文字列を比較する

それぞれの詳しい使い方は、p.327〜p.336を参照して下さい

文字列操作の例

【例9】

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

int main(void)
{
    char *str9 = "Hello";
    int len;
    len = strlen(str9);
    printf("文字列の長さ: %d\n", len);

    return 0;
}
```

文字列に関する標準ライブラリ関数を使う時にインクルードする

関数strlen()は、引数が参照している文字列の長さを返す

【練習5-1】

例1と例2にある処理をmain()内に書いて、プログラムを実行し、配列による文字列とポインタによる文字列を出力してみましょう。

【練習5-2】

例7と例8にある処理をmain()内に書いて、プログラムを実行し、文字列の配列を出力してみましょう。

【課題5-1】

まず、次のような文字列に対するポインタstrを
main()内に宣言します。

```
char *str1 = "HappyHalloween";
```

この文字列の文字の間にハイフン（-）を出力するプログラムをmain()に作成してください。

- 文字列は「文字の配列」であることに着目して、例2のポインタstr2を参考に、「str2+i」の繰り返し処理を作る
  （次のスライドのmainの繰り返し処理を参照）
- 繰り返し処理の中では、文字とハイフンをセットで出力する

【課題5-1】

[mainでの処理]

```
int main(void) {
  char *str1 = “HappyHalloween";
  int i

  /* strの最初から末尾まで繰り返す */

  for(i=0; *(str1+i)!='\0'; i++) {
    /* ここでstrのi番目の文字とハイフンを一緒に出力する */

  }
  return 0;
}
```

str1が参照している文字列に対する繰り返し処理

[実行結果]

```
 H-a-p-p-y-H-a-l-l-o-w-e-e-n-
```

【課題5-2】

課題5-1のプログラムを参考に、仮引数が参照している文字列に対して、仮引数で指定した文字で区切られた文字列を出力する関数show_separation()を作成してください。

[この関数のプロトタイプ宣言]

```
void show_separation(char *s, char ch);

  /* 文字列sに対する繰り返し処理を作る */
  /* s+iが参照する文字とchをセットで出力する */
```

【課題5-2】

[mainでの処理]

```
char *str0 = "TrickOrTreat";
char *str1 = "HappyHalloween";
show_separation(str0, '|');
show_separation(str1, '*');
```

[実行結果]

```
T|r|i|c|k|O|r|T|r|e|a|t|
H*a*p*p*y*H*a*l*l*o*w*e*e*n*
```

【課題5-3】

仮引数が参照している文字列に対して、仮引数で指定した文字の置換をする関数replace_char()を作成してください。

[この関数のプロトタイプ宣言]

```
void replace_char(char *s, char ch1, char ch2);

  /* 文字列sに対する繰り返し処理を作る */
  /* s+iが参照する文字がch1である場合、ch2に置き換える */
  /* 関数内で文字列sを出力するのではなく、
        置換した結果が文字列sに反映されるように作る*/
```

【課題5-3】

[mainでの処理]

```
char str2[100] = "TrickOrTreat";
char str3[100] = "HappyHalloween";
replace_char(str2, 'r', '_');
printf("str2: %s\n", str2);
replace_char(str3, 'e', 'E');
printf("str3: %s\n", str3);
```

[実行結果]

```
str2: T_ickO_T_eat
str3: HappyHallowEEn
```

【課題5-4】

仮引数が参照している文字列の配列に対して、各文字列を出力する関数show_strarray()を作成してください。

[この関数のプロトタイプ宣言]

```
void show_strarray(char *sarray[]);

  /* 例8に示した文字列の配列の出力処理を参考に、
        この処理を関数内に作る */
```

【課題5-4】

[mainでの処理]

```
 char *keyword[5] = { "variable", "function", "array",
                      "pointer", "string" };
 show_strarray(keyword);
```

[実行結果]

```
 *(sarray+0): variable
 *(sarray+1): function
 *(sarray+2): array
 *(sarray+3): pointer
 *(sarray+4): string
```

まだ余裕のある人は…

【課題5-5】

仮引数が参照している2つの文字列に対して、これらの文字列の長さを比較し、その結果を返す関数compare_strlen()を作成してください。

[この関数のプロトタイプ宣言]

```
int compare_strlen(char *s1, char *s2);

  /* 関数strlen()を使って文字列s1とs2の長さをそれぞれ求める */
  /* これらの長さを比較し、その結果に応じて以下のような整数値を返す */
  /*    ● s1の方が長い文字列である場合は、1を返す */
  /*    ● s2の方が長い文字列である場合は、-1を返す */
  /*    ● s1とs2が同じ長さの文字列である場合は、0を返す */
```

【課題5-5】

[mainでの処理]

```
printf("compare_strlen: %d\n",
       compare_strlen(str0, str1));
printf("compare_strlen: %d\n",
       compare_strlen("Google", "Amazon"));
printf("compare_strlen: %d\n",
       compare_strlen("Microsoft", "Apple"));
```

[実行結果]

```
compare_strlen: -1
compare_strlen: 0
compare_strlen: 1
```

小テストについて

小テストの注意点

- □ 他人の力は借りずに、自分だけでプログラムを作成する。（つまり定期試験と同様）
- □ 小テスト中は、演習室外へのネットワークアクセスは遮断される。

小テストについて

小テスト中に参照できるもの

- □ 教科書, 配付資料
- □ 自分のホームディレクトリ（ホームフォルダ）以下に保存されているファイル
- □ 小テストでは紙媒体のものは参照可能
- □ 上記以外の情報を参照することは不正行為とする
  例：USBで接続された機器に保存されているファイルの参照
  ネットワークを介した情報の参照、など