情報システムプログラミングII（14回目）

2024年7月23日（火）
5～6限（授業振替）

授業内容

- 講義内容（教科書の417～438ページ＋α）
  - 文字列を扱う標準関数
  - 文字列の配列
  - Pythonにおける文字列の扱い
- 前期末試験の説明
- 演習課題

文字列を扱う標準関数

■strlen関数

- 文字列の長さ（バイト数）を取得する
- strlen関数を利用するための構文
  - ➢ string.hの読み込み（#include <string.h>）が必要

size_t strlen(const char* str);

str　　：文字列として確保したメモリ領域の先頭アドレス

戻り値：先頭から ¥0 までのバイト数（ただし ¥0 は含まない）

ヌル文字（¥0）は含まない

日本語などのマルチバイト文字は想定されておらず，正しい文字数は取得不可（バイト数は取得可）

文字列を扱う標準関数

■strlen関数

- strlen関数の利用例

```
int main(void)
{
  char str[1024] = "c language";
  int len = strlen(str);
  printf("%d¥n", len);

  return 0;
}
```

確保したメモリは 1024 バイト

実行結果

```
10
```

size_t型は0以上の整数でunsigned int型やunsigned long型にキャストされるため，厳密には%uや%luが正しい

文字列を扱う標準関数

■strcmp関数

- 文字列を比較する（同じかどうか判断する）
- strcmp関数を利用するための構文
  - ➢ string.hの読み込み（#include <string.h>）が必要

int strcmp (const char* str1, const char* str2);

str1　：「文字列として利用」しているメモリ領域の先頭アドレス
str2　：「文字列として利用」しているメモリ領域の先頭アドレス
戻り値：2つのメモリ領域が「文字列として」等しければ0

同じでない場合，1つ目の文字列が2つ目の文字列よりも
文字コードが前の場合は負の値を，後ろの場合は正の値を返す

文字列を扱う標準関数

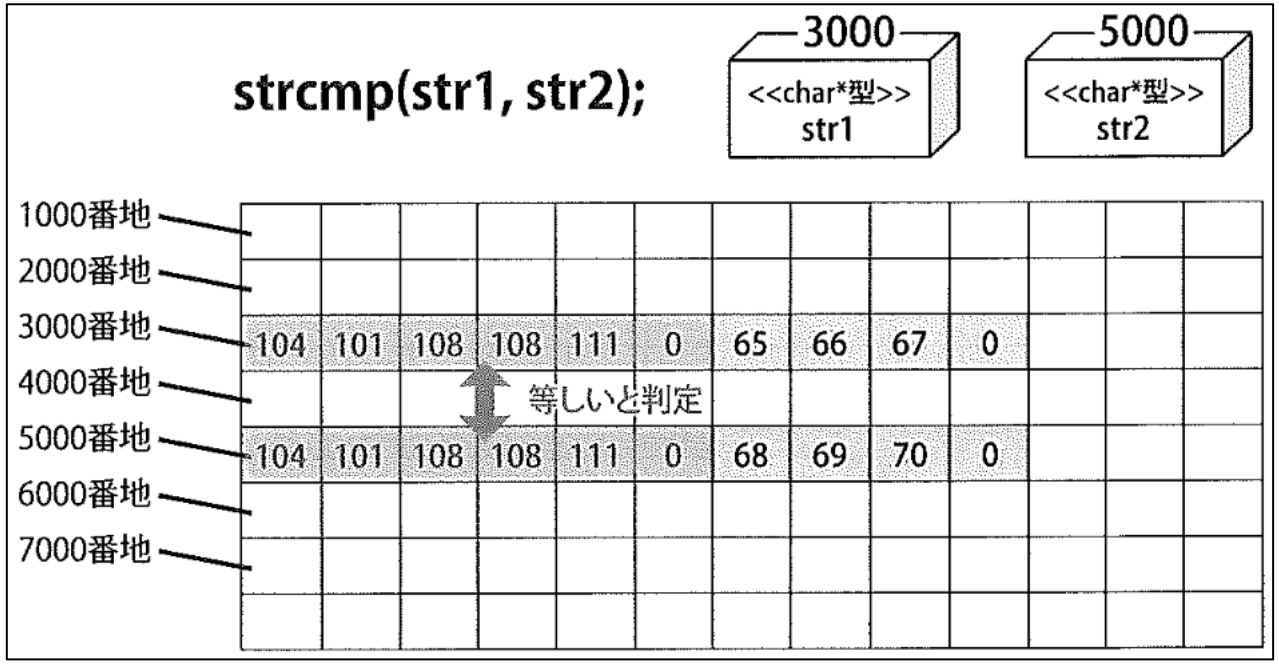
■strcmp関数

- strcmp関数の利用例

```
char str1[] = "hello¥0ABC";
char str2[] = "hello¥0DEF";
if (strcmp(str1, str2) == 0) {
  printf("文字列として等しい");
}
```

仮に 3000 ～ 3005 番地

仮に 5000 ～ 5005 番地



文字列（指定したアドレスからヌル文字まで）を比較したい場合はstrcmp関数，
純粋にメモリ領域を比較したい場合は
memcmp関数などと使い分けする

文字列を扱う標準関数

■strcpy関数

- 文字列をコピーする
- strcpy関数を利用するための構文
  - ➢ string.hの読み込み（#include <string.h>）が必要

char* strcpy (char* dest, const char* src);

dest　：コピー先のメモリ領域の先頭アドレス（十分なメモリ領域が確保済みであること）

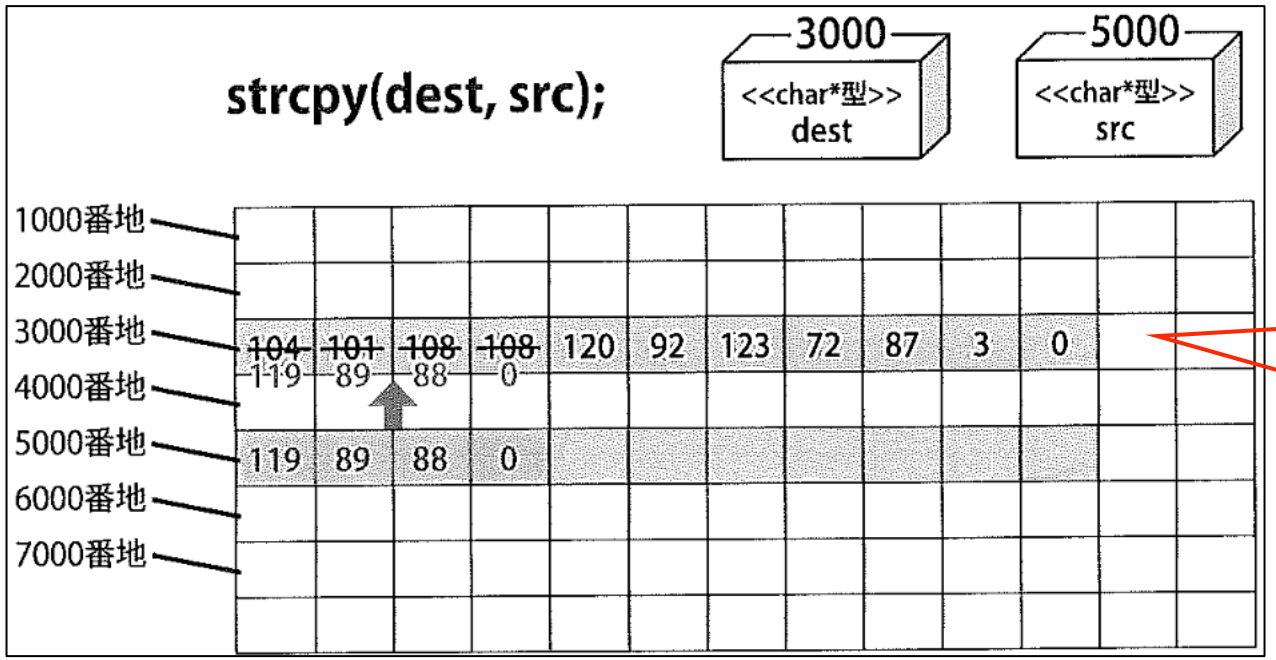
src　：コピー元文字列が格納されている先頭アドレス

戻り値：dest と同じ

文字列を扱う標準関数

■strcpy関数

- strcpy関数の利用例



コピー先のメモリ領域を十分に確保しないとオーバーランすることになるので注意！

文字列を扱う標準関数

■strcat関数

- 文字列を連結する
- strcat関数を利用するための構文
  - ➢ string.hの読み込み（#include <string.h>）が必要

char* strcat (char* dest, const char* src);

dest　：連結先のメモリ領域の先頭アドレス（現状の ¥0 より後ろに src を連結できるよう十分なメモリ領域が確保済みであること）

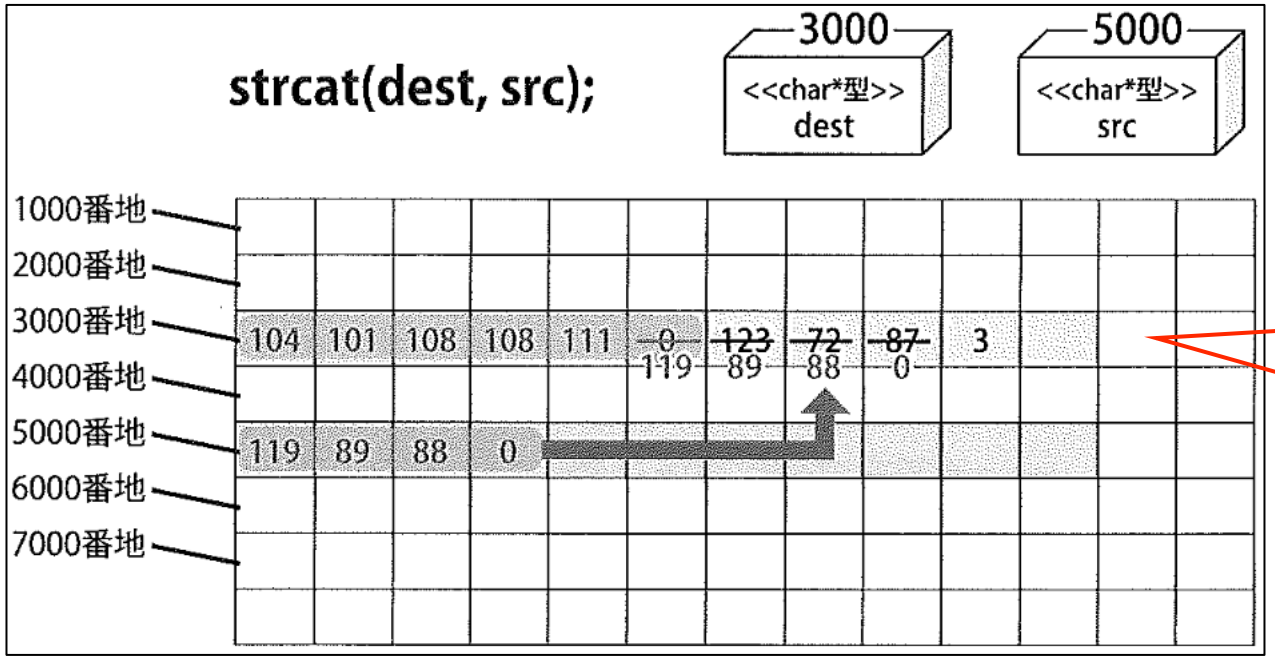
src　：連結したい文字列が格納されている先頭アドレス

戻り値：dest と同じ

文字列を扱う標準関数

■strcat関数

- strcat関数の利用例



連結先のメモリ領域を十分に
確保しないとオーバーランする
ことになるので注意！

文字列を扱う標準関数

■sprintf関数

- 書式を指定して文字列をメモリ領域に出力する
- sprintf関数を利用するための構文

int sprintf(char* dest, const char* format, …);

dest　：結果を書き込むメモリ領域の先頭アドレス（あらかじめ十分な容量を確保してあること）

format：printf と同様の書式文字列

…　：書式文字列中のプレースホルダに対応した値

戻り値：成功した場合は dest に書き込んだ文字数（¥0 を含まない）
失敗した場合は負の数

文字列を扱う標準関数

■scanf関数

- 書式を指定してキーボードからの入力を受け付ける
- scanf関数を利用するための構文

int scanf(const char* format, …);

format：入力用の書式文字列（詳細は付録 E.8.2）

…　　：書式文字列中のプレースホルダに対応した値を格納するためのメモリ領域の先頭アドレス（各領域には十分なメモリ容量を確保してあること）

戻り値：成功した場合は読み込んだ項目の個数、失敗した場合は EOF

文字列を扱う標準関数

■scanf関数

- scanf関数の利用例

```
int main(void)
{
  char name[1024];
  int hp;
  printf("名前とHPをスペース区切りで入力してください。¥n");
  scanf("%s %d", name, &hp);
  printf("入力された名前:%s¥n入力されたHP:%d¥n", name, hp);

  return 0;
}
```

第2引数としてアドレスを指定する必要があるので，配列名はそのままで（先頭アドレスを表すため），変数名には&を付ける

文字列を扱う標準関数

■より安全な標準関数

- オーバーラン防止やエラー処理など，従来の関数をより安全に利用可能な代替関数がC言語の規格には存在する
- 以下は代替関数の一例だが，実装環境（GCCなど）によっては利用できない

fprintf_s	fscanf_s	gets_s	localtime_s
memcpy_s	printf_s	scanf_s	snprintf_s
sprintf_s	sscanf_s	strcat_s	strcpy_s
strncat_s	strncpy_s	strnlen_s	

文字列の配列

■コマンドライン引数

- プログラムの実行時にプログラムに渡す文字列のこと
  - ➢ 半角スペースで区切って指定する
- main関数の引数を以下のように指定することで（変数名は任意だが型は固定），main関数内でコマンドライン引数を利用可能

```
int main(int argc, char** argv)
{
   ⋮
}
```

```
$ ./puzmon ミサキ 1000⏎
パズモンRPG  v1.0
ミサキ（HP=1000）でゲームを開始します！
```

文字列の配列

■コマンドライン引数

- int型の第1引数で，コマンドライン引数の数+1を取得可能

```
int main(int argc, char** argv)
{
  printf("argc=%d¥n", argc);
  return 0;
}
```

実行結果

```
$ ./a.out⏎
argc=1
$ ./a.out hello C⏎
argc=3
```

実行可能ファイル（a.out）を実行

「+1」されることに注意！

文字列の配列

「文字列として」なので，
数値として利用したい場合は
プログラム中で変換が必要

■コマンドライン引数

- char**型の第2引数で，コマンドライン引数自体を取得可能
  - ➢「**」の付く型は，ポインタ型のアドレスを格納するためのポインタ型（二重ポインタ）

argvには，各コマンドライン引数が格納されているメモリ領域の先頭アドレスが順に格納されている

```
int main(int argc, char** argv)
{
  printf("argc=%d¥n", argc);
  for (int i = 0; i < argc; i++) {
    char* strAddr = argv[i];
    printf("%d番目の情報: %s¥n", i, strAddr);
  }
  return 0;
}
```

i番目の文字列の先頭アドレスを取得

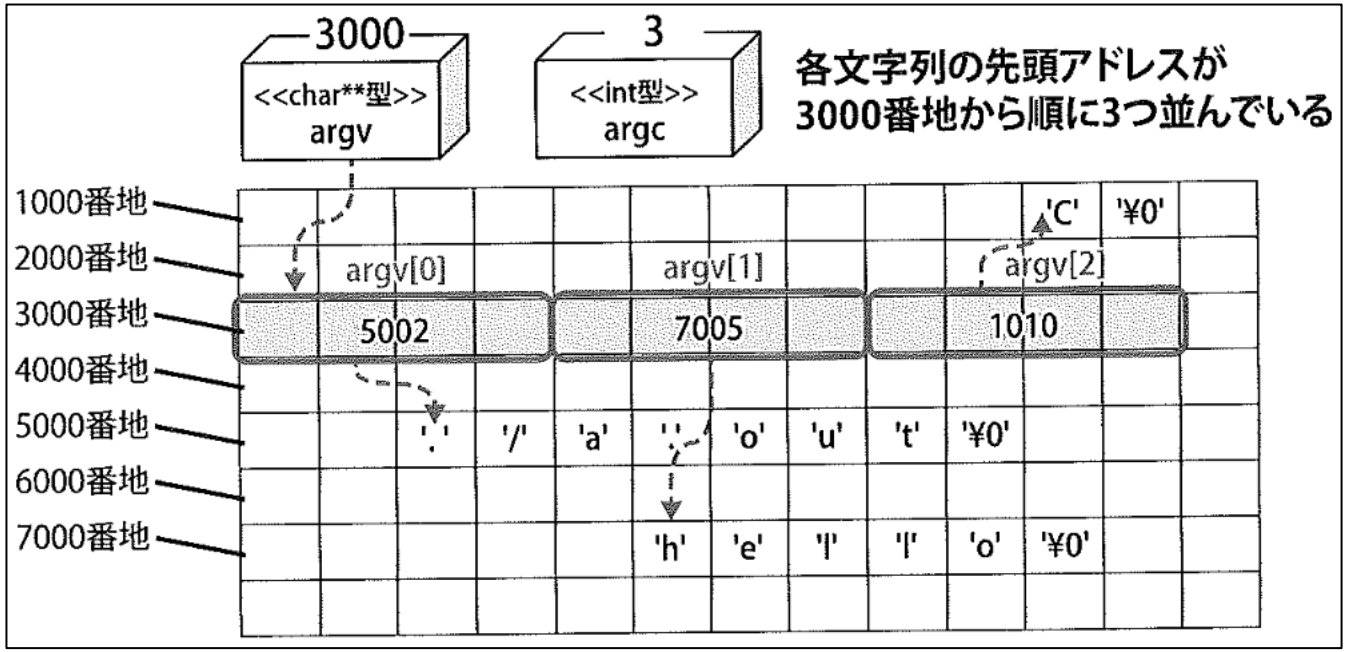
実行結果

```
$ ./a.out⏎
argc=1
0番目の情報: ./a.out
$ ./a.out hello C world⏎
argc=4
0番目の情報: ./a.out
1番目の情報: hello
2番目の情報: C
3番目の情報: world
```

文字列の配列

■コマンドライン引数

- char**型の第2引数で，コマンドライン引数自体を取得可能



文字列の配列

■複数の文字列を扱う方法

- 配列の配列

はじめの要素数は省略可能
（この場合はstr[][21]でもよい）

```
char str[4][21] = {"Tottori", "Kurayoshi", "Yonago", "Sakaiminato"};
for(int i=0; i<4; i++){
    printf("%d個目の文字列：%s¥n", i+1, str[i]);
}
```

- ポインタの配列（二重ポインタ）

はじめの要素数は省略可能
（この場合はstr[]でもよい）

```
char* str[4] = {"Tottori", "Kurayoshi", "Yonago", "Sakaiminato"};
for(int i=0; i<4; i++){
    printf("%d個目の文字列：%s¥n", i+1, *(str+i));
}
```

Pythonにおける文字列の扱い

■Pythonにおける文字数の取得：len関数

- 引数は文字列，戻り値は文字数

```
len1 = len("こんにちは")
print(len1)
len2 = len("Hello")
print(len2)
```

日本語などのマルチバイト文字も
正しく文字数を取得できる

■Pythonにおける文字列の比較

- 完全一致（大小文字の区別含む）は「==」か「!=」で判定可能
- 大小関係（文字コードでの順序）は「<」，「<=」，「>」，「>=」で判定可能

Pythonにおける文字列の扱い

■Pythonにおける文字列のコピー

- 様々な方法があるが，メモリ領域において同じアドレスを指すのか，異なるアドレスにコピーした上でそこを指すのか，方法によって異なるので注意

```
str1 = "Hello"
str2 = "" + str1
print(str2)
str3 = str(str1)
print(str3)
```

この方法はどちらもsrt1の文字列をコピーするもので，異なるアドレスにコピーした上でそこを指している
（str1の文字列を変更してもstr2とstr3の文字列は変わらない）

Pythonにおける文字列の扱い

■Pythonにおける文字列の連結：「+」の利用

- 変数と文字列リテラルが混在しても連結できる

```
str1 = "Hello"
str2 = str1 + " World!"
print(str2)
str3 = "Hello" + " " "World" + "!"
print(str3)
```

■Pythonにおける文字列に係る処理

- 文字列の分割，文字種の判定，文字列の置換などの様々な処理を，既存の関数やメソッドで実現可能