情報システムプログラミングⅡ（13回目）

2024年7月19日（金）

3～4限

授業内容

- 講義内容（教科書の396～416ページ）
  - 文字列の扱い
  - 文字列リテラル
  - 文字列の受け渡し
  - 文字列の実装と注意点
  - 文字列とオーバーラン
- 演習課題

文字列の扱い

■char型

- 一文字を格納するための型
- char型の連続した領域を確保することで文字列を格納できる

1024個の要素を有するchar型の配列にStringという型名を付与

```
typedef char String[1024];

int main(void)
{
  char ages[1024] = {19, 21, 29, 29};
  String str = "hello";
  printf("%s¥n", str);

  return 0;
}
```

仮に 4000 ～ 5023 番地

仮に 8000 ～ 9023 番地

agesとstrは
どちらも同じ型と
サイズの領域を
持つ変数

文字列の扱い

■終端文字／ヌル文字（¥0）

- 文字列の末尾にある「¥0」により文字列の終わりを判定

```
int main(void)
{
  char ages[1024] = {19, 21, 29, 29};
  String str = "hello";
  printf("%s¥n", str);

  for (int i = 0; i < 10; i++) {
    printf("%d, ", str[i]);
  }

  return 0;
}
```

仮に 4000 ～ 5023 番地

仮に 8000 ～ 9023 番地

ヌル文字

実行結果

```
hello
104, 101, 108, 108, 111, 0, 24, 82, 124, 41
```

実行する環境によって変わる

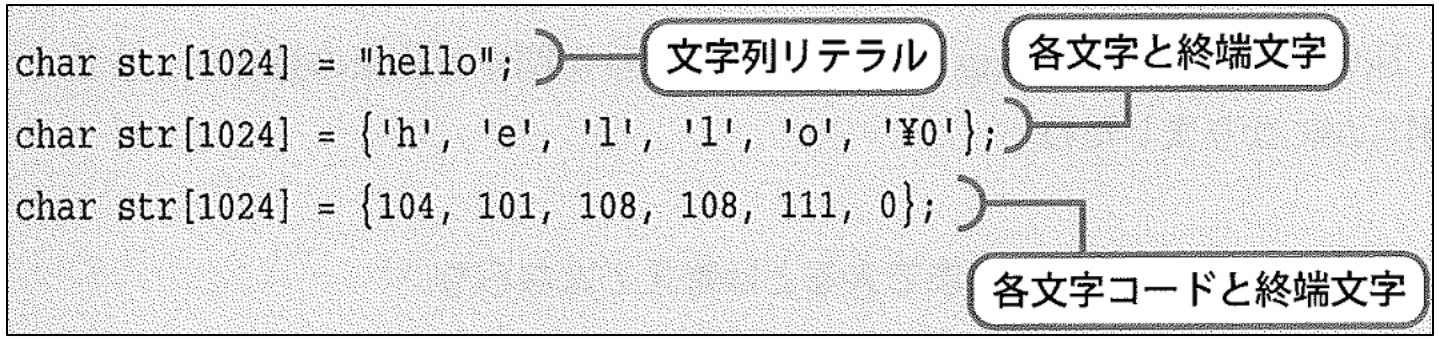
文字列リテラル

■文字列リテラルとは

- 「"」で囲まれた文字列情報のことで，末尾には自動的に「¥0」が付与される

これらは全て
同じ意味（処理）

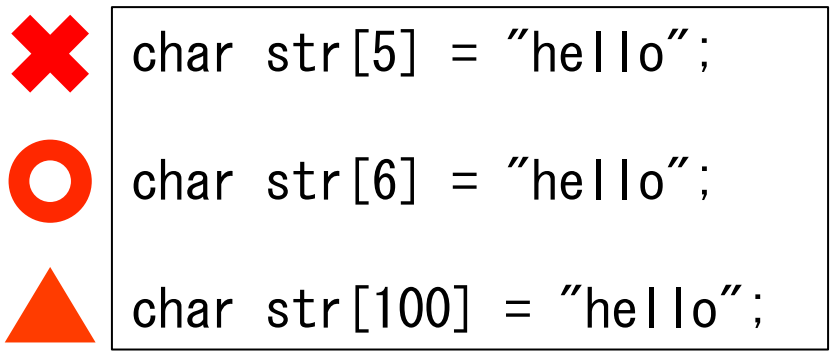
一文字の場合は
「'」で囲う



文字列リテラル

■文字列の格納で多い間違い

- ヌル文字を格納するための領域が確保されていない
  - ➢ 文字の数にヌル文字分（1）を加えた領域が必要となる



文字列リテラル

■要素数を省略した文字列によるchar型配列の初期化

- char型配列の定義時に要素数を省略することで，文字列の格納に必要なメモリ領域のサイズが自動的に設定される
- 要素数を省略して文字列を定義（格納）する構文

char str[ ] = " 初期値 ";

※要素数の指定を省略することで、「¥0 を含む必要な要素数」が自動的に設定される。

```
char str[] = "hello";
char longstr[] = "helloworld";
memcpy(str, longstr, 11);
```

6 バイトぴったりで安全に確保されるが…

長い文字列で上書きしてしまうとオーバーランが発生

文字列の受け渡し

■printf関数の挙動

- 文字列を表示する場合，引数として渡されたアドレスからメモリ領域に格納された情報を順に表示し，ヌル文字に到達すると表示を終える
  - ➢ 文字列を表示する際は，その先頭アドレスだけを渡せばよい

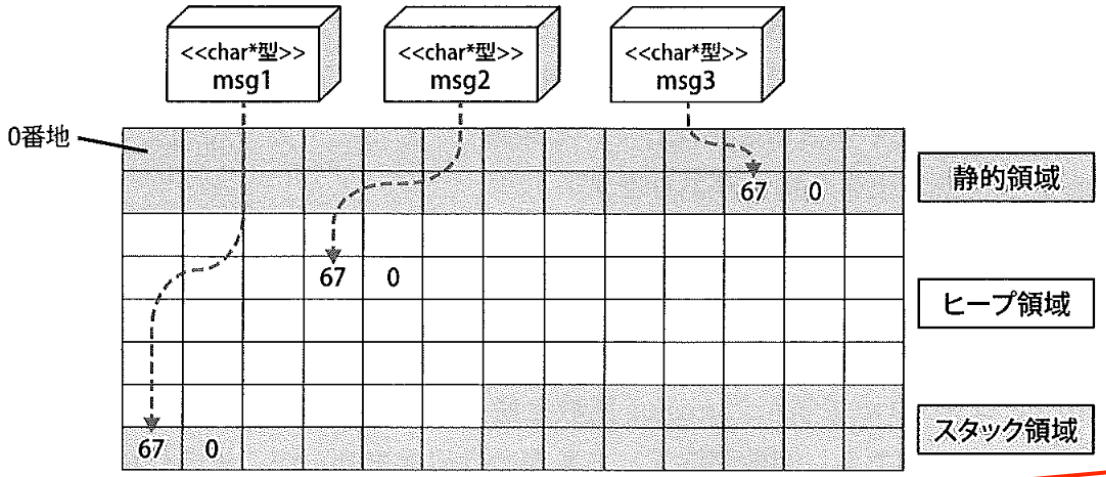
```
char str[1024] = "hello";
printf("%s", str);
```

```
char str[1024] = "hello";
char* p = str;
printf("%s, %s", str, p);     // どちらも同じように表示できる
```

文字列の実装と注意点

■文字列を格納するためのメモリ領域の確保

- 連続したメモリ領域を確保すればよく，char型の配列以外の方法も利用できる



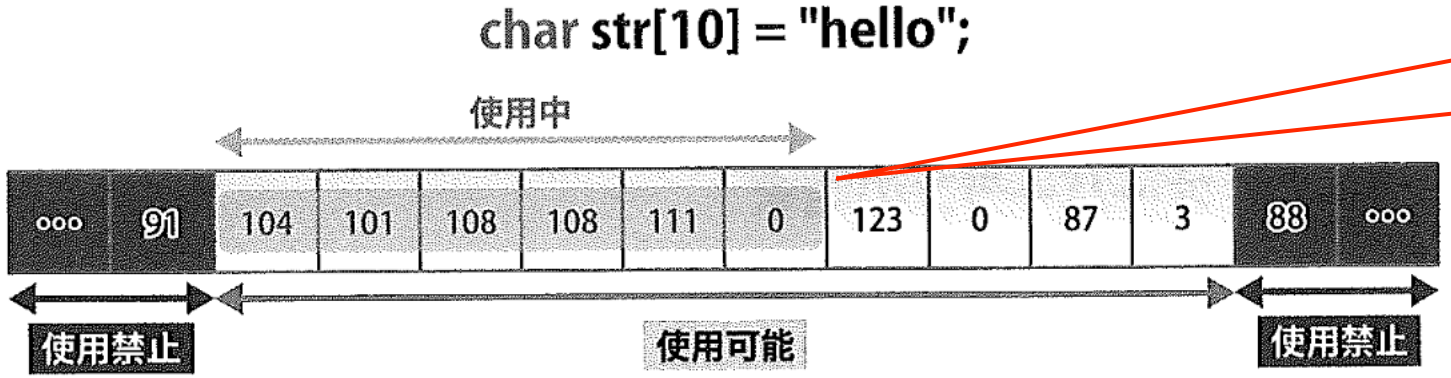

```c
int main(void)
{
  // 手段①
  char array[1024] = "C";          // 配列宣言でメモリを確保
  char* msg1 = array;              // からくり構文②で先頭アドレス取得
  printf("%s", msg1);              // printf("%s", array)でも同じ意味

  // 手段②
  char* msg2 = (char*)malloc(1024); // malloc でメモリを確保
  msg2[0]= 'C';
  msg2[1] = '¥0';
  printf("%s", msg2);
  free(msg2);                      // 確保したメモリの解放

  // 手段③
  const char* msg3 = "C";          // リテラルでメモリを確保
  printf("%s", msg3);

  return 0;
}
```

静的領域に格納された情報の先頭アドレスが渡される，
constは変更不可とするため付与（必須ではないが推奨）

文字列とオーバーラン

■ヌル文字の重要性

- 何らかの原因でヌル文字が無くなると文字列の終わりが判断できず，後ろに続くメモリ領域をヌル文字が表れるまで読み取り続けることになる
- 何らかの原因で文字列中にヌル文字が加えられると，文字列の途中で読み込みが終了することになる



数値や文字の「0」とヌル文字（¥0）は異なるので注意！