情報処理実習

第12回：文字列

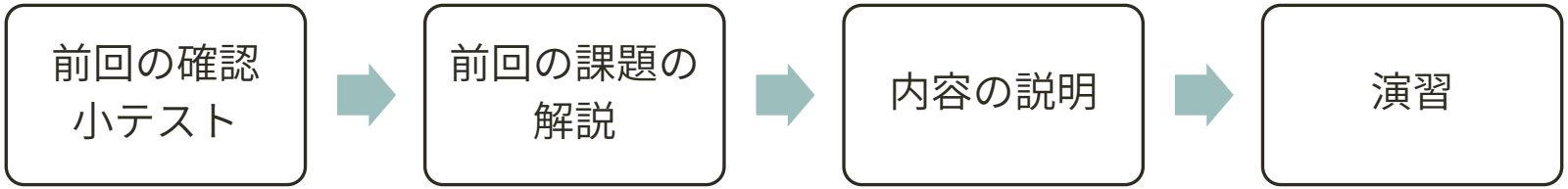
2017年12月11日(月)

担当：佐藤

アナウンス

- 授業前にやること
  - 計算機にログオン
  - CoursePowerにログイン
- 注意事項
  - 教室内飲食禁止
  - 原則として，授業中ではなく休み時間にトイレにいきましょう

授業の進め方(確認)



基本事項の確認小テスト

① CoursePowerにログインして，すぐに解答が始められるよう準備する
② 開始の合図があるまで，解答を始めずに待機する

注意

「閉じる」ボタンは決して押さないこと！

前回課題の解説

課題1(1)
教科書に準じた解答

```c
#include <stdio.h>

int main() {
   int c;

   printf("1文字入力してください¥n");
   c = getchar();
   if (97 <= c && c <= 122) {
      putchar((char)(c - 32));
   } else if (65 <= c && c <= 90) {
      putchar((char)(c + 32));
   }
   printf("¥n");

   return 0;
}
```

getchar()の戻り値の型はint

getchar()は戻り値を代入することで変数に文字を設定する．scanf()との違いに注意！

cはintなので文字コードで比較

文字コードから文字に変換するためint→charへキャスト

- アルファベット小文字に対応するASCIIコードは97以上122以下
- アルファベット大文字に対応するASCIIコードは65以上90以下

課題1(2)
別解

```
#include <stdio.h>

int main() {
   int c, d = 'a' - 'A';

   printf("1文字入力してください¥n");
   c = getchar();
   if ('a' <= c && c <= 'z') {
      putchar(c - d);
   } else if ('A' <= c && c <= 'Z') {
      putchar(c + d);
   }
   putchar('¥n');

   return 0;
}
```

「小文字の文字コード>大文字の文字コード」であることを前提として，辞書式順序が対応する小文字と大文字の差をあらかじめ変数にしておく．順番が対応していれば文字の組み合わせはなんでもよい(例: 'b' - 'B')

putchar()は実引数を文字コードとして解釈するため，実引数はcharへキャストせずintのまま渡して構わない．なお，int⇄charのキャストにより値は変わらない

文字定数は文字コードとして解釈されるため，文字コードの代わりに文字定数を書けばよい(**文字に対応する文字コードを覚えておく必要はない！**)

課題2(1)
教科書に準じた解答

scanf()で文字を入力するためchar

```
#include <stdio.h>

int main() {
   char c;

   printf("1文字入力してください¥n");
   scanf("%c", c);
   if ((char)48 <= c && c <= (char)57) {
      printf("%d¥n", (int)c - 48);
   }

   return 0;
}
```

文字を読み込む場合の変換指定子は%c

ASCIIコード表における文字位置の前後関係で比較

文字を文字コードにするためintへキャスト

- 数字に対応するASCIIコードは48以上57以下
- **数値 ＝ 数字のアスキーコード - 数字0のアスキーコード**

整数(アスキーコードが整数なので)

課題2(2)
別解

```
#include <stdio.h>

int main() {
   char c, d = '0';

   printf("1文字入力してください¥n");
   scanf("%c", c);
   if ('0' <= c && c <= '9') {
      printf("%d¥n", c - d);
   }

   return 0;
}
```

数字と数値の差をあらかじめ変数にしておく．charは整数型なので，intで宣言する必要はない

%d変換指定子はデータ型を問わないため値のみ注意すればよい．int⇄charのキャストにより値は変わらないためintへのキャストは不要．仮にintへキャストした場合，プログラムに余計な処理をさせることになるだけでなく，メモリ上の領域を余計に使用することにもなる

課題3

```
#include <stdio.h>

int main() {
   int c;

   c = getchar();
   do {
      if ('a' <= c && c <= 'z' || 'A' <= c && c <= 'Z'
                               || c == ' '
                               || c == '¥n') {
         putchar((char)c);
      } else {
         putchar('?');
      }
   } while((c = getchar()) != EOF);

   return 0;
}
```

(char)不要

空白文字と改行文字を考慮しないと表示が問題文の実行結果と一致しない

getchar()による文字の読み込みを繰り返す場合のお決まりの書き方.
「c = getchar()」を()で囲むことを忘れないよう注意!

課題4

```c
#include <stdio.h>

int main() {
   int i;

   printf("   |");
   for (i = 0; i <= 9; i++) {
      printf("%3d|", i);
   }
   printf("¥n");
   for (i = 0; i <= 10; i++) {
      printf("----");
   }
   printf("¥n");

(右へ続く)
```

```c
   for (i = 0; i <= 127; i++) {
      if (i == 0) {
         printf("%3d|", i);
      } else if (i % 10 == 0) {
         printf("¥n%3d|", i);
      }
      if (33 <= i && i <= 126) {
         printf("  %c ", (char)i);
      } else {
         printf("    ");
      }
   }
   printf("¥n");

   return 0;
}
```

ループ変数を文字コードとして扱う点がポイント！

目に見える文字だけを表示

(char)不要

目に見えない文字は空白にする

文字列の概要

文字列

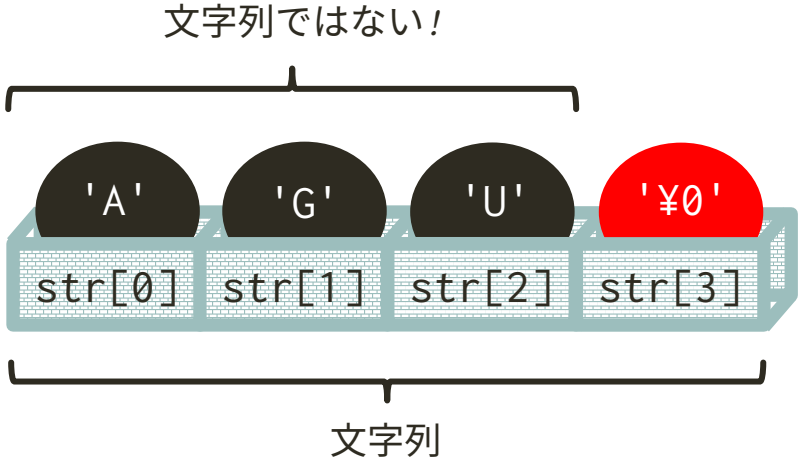
- 最後の要素が'¥0'(終端文字，ナル文字，ヌル文字)の文字の配列(文字配列)を文字列という

例

ヌル文字分も考慮！

```
char str[4];


str[0] = 'A';
str[1] = 'G';
str[2] = 'U';
str[3] = '¥0';
```



文字配列の初期化

- "(二重引用符，ダブルクォーテーション)で囲まれた文字の並びを文字列定数(文字列リテラル)という
  - 文字列定数は，**末尾にヌル文字が自動で付加**される
- 文字列定数を用いて文字配列を初期化することができる

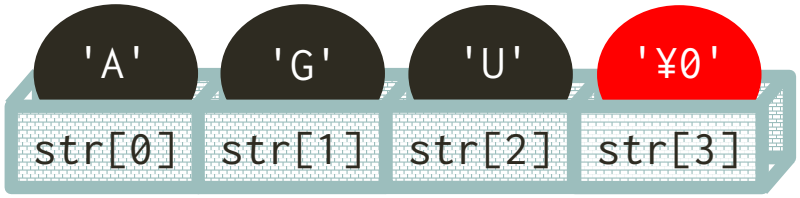
文法

```
char 文字配列名[要素数] = 文字列定数;
```

例

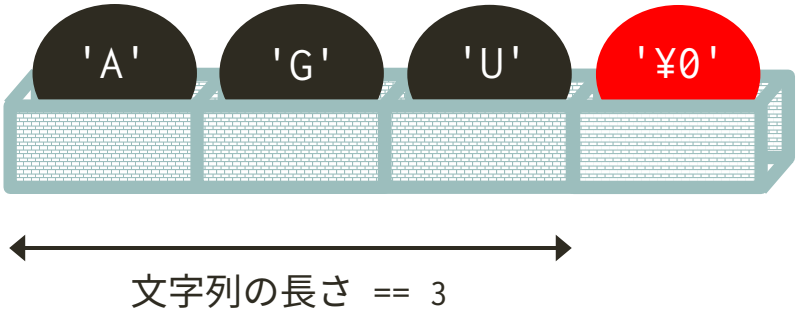
```
char str[4] = "AGU";
```

ヌル文字は書かなくてよい！



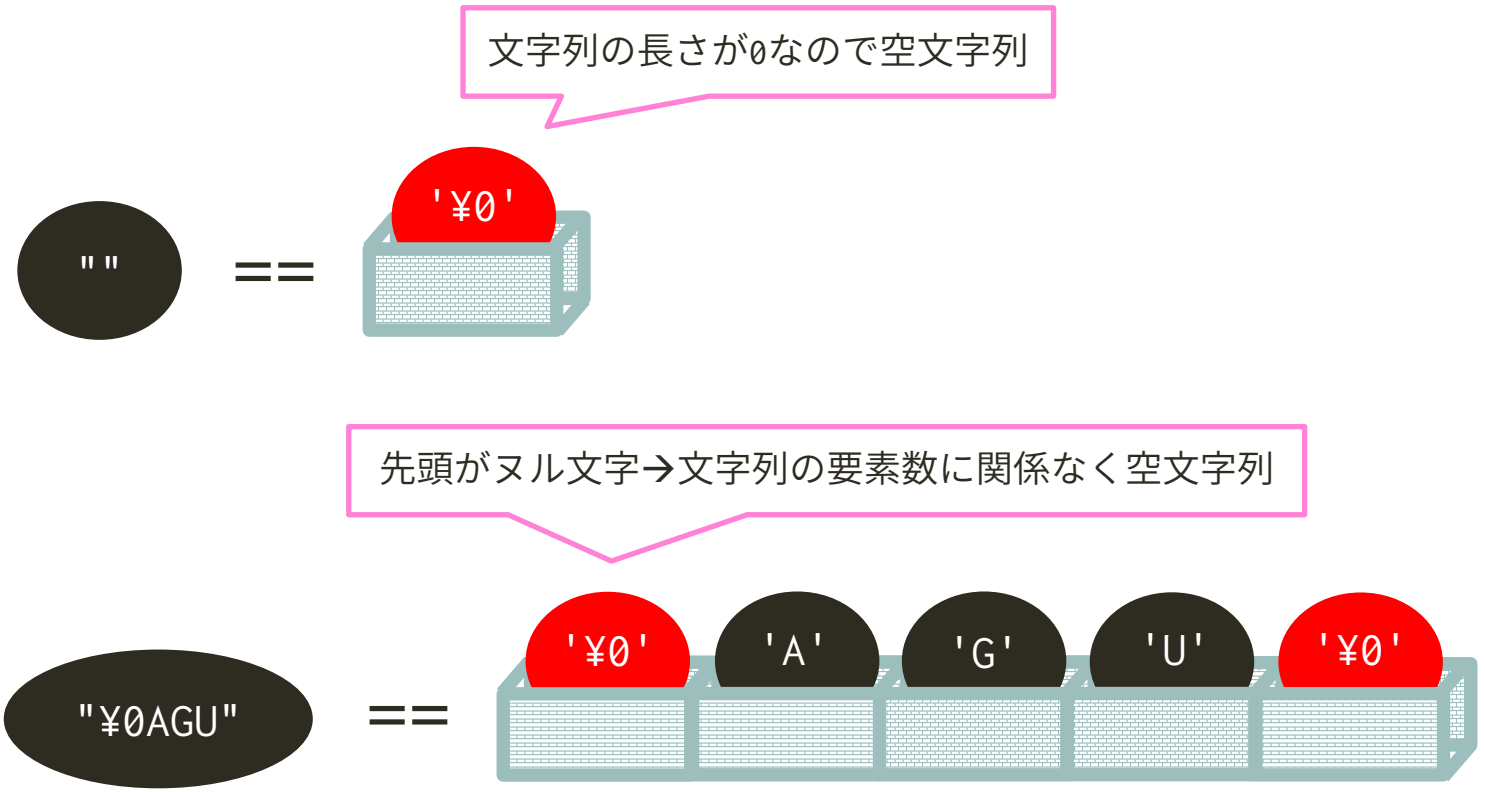
文字列の長さ

- 文字列の先頭の文字からヌル文字まで(ヌル文字を除く)の文字の個数を**文字列の長さ**という
- **文字列の要素数 ＝ 文字列の長さ ＋ 1**
- 文字配列を宣言する際，配列の要素数には文字列の長さではなく文字列の要素数を設定すること



空文字列

- 文字列の長さが0の文字列を空(から，くう)文字列という



文字列の入出力

文字列の入力(1)

- scanf()を使った文字列の入力

文法

&は不要！

```
scanf("%s", 文字配列名);
```

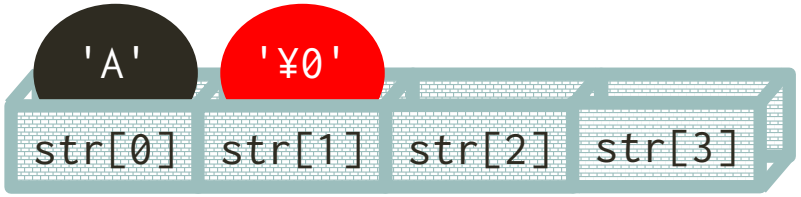
- 空白類文字(※)に出会うと読み込みを終了し，**最後にヌル文字('¥0')を付加する**(空白類文字は読み込まれない)

例

ヌル文字分も考慮！

```
char str[4];


scanf("%s", str); /* 「A␣U」を入力 */
```



※空白文字('␣')，改行文字('¥n')，タブ文字('¥t')などの総称

文字列の入力(2)

- gets()を使った文字列の入力

文法

&は不要！

```
gets(文字配列名);
```

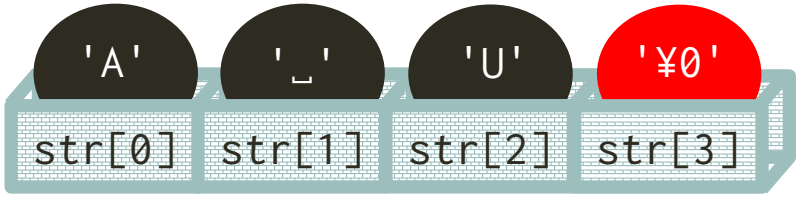
- 改行文字('¥n')に出会うと読み込みを終了し，最後にヌル文字('¥0')を付加する('¥n'は読み込まれない)
- **空白文字('␣')を読み込める！**

例

ヌル文字分も考慮！

```
char str[4];


gets(str); /* 「A␣U」を入力 */
```



NULL

- 入力終了を表す文字コード^Zが入力された場合のgets()の戻り値は**NULL(ナル，ヌル)**
  - コマンドプロンプトで^Zを入力するにはCtrl+Zを押す

例

```
char str[4];


if (gets(str) != NULL) {
   printf("Yes¥n");
else {
   printf("No¥n");
}
```

実行結果

```
AGU
Yes
```

実行結果

```
^Z
No
```

文字列の出力(1)

- printf()を使った文字列の出力

文法

文字配列名 or 文字列定数

```
printf("%s", 文字列);
```

  - 文字列の先頭の文字からヌル文字('¥0')の1つ前の文字まで出力

例

```
char str[4] = "AGU";


printf("%s¥n", str);
printf("%s¥n", "AGU");
```

printf("AGU¥n");と同じ

実行結果

```
AGU
AGU
```

文字列の出力(2)

- puts()を使った文字列の出力

文法

```
puts(文字列);
```

- 文字列の先頭の文字からヌル文字('¥0')の1つ前の文字まで出力し，**最後に改行文字('¥n')を付加**する

例

```
char str[4] = "AGU";


puts(str);
puts("AGU");
```

実行結果

```
AGU
AGU
```

printf()と異なり，'¥n'を書かなくても勝手に改行される

文字列入力の詳細

- エンターキー(リターンキー)を押すと2つの効果がある
  1. **改行文字('¥n')の入力**
  2. 入力内容の確定
  - **②だけでなく①の効果もあることに注意!**

例

```
#include <stdio.h>

int main() {
   char str1[6], str2[6];

   gets(str1);           /* "A G U"を入力 */
   puts(str1);           /* "A G U"を出力 */
   scanf("%s", str2);    /* "A G U"を入力 */
   printf("%s¥n", str2); /* "A"を出力 */

   return 0;
}
```

gets()は'A'→'␣'→'G'→'␣'→'U'と先頭から順に文字をstr1に代入し，最後の文字である'¥n'に出会うと'¥n'の代わりに'¥0'を代入して読み込みを終える．したがって，str1には，「'A''␣''G''␣''U''¥0'=="AGU"」という文字列が設定される

gets()とscanf()の読み込み対象は「'A''␣''G''␣''U''¥n'」という文字配列(末尾が'¥0'ではないため，この時点では文字列ではない)

scanf()もgets()同様，先頭から順に文字を(str2に)代入していくが，こちらは空白文字('␣')を読み込めないため'A'の次に'␣'に出会うと読み込みを終え，'␣'の代わりに'¥0'を代入する．したがって，str2には，「'A''¥0'=="A"」という文字列が設定される

文字列操作関数

string.h

- 標準ヘッダファイルのひとつstring.hをインクルードしておくことにより，あらかじめ用意された便利な文字列操作関数が使えるようになる
  - #include <string.h>
- 代表的な文字列操作関数は次の通り

関数	関数の機能
strcpy() (ストリングコピー)	文字列の代入
strcat() (ストリングキャット)	文字列の連結
strcmp() (ストリングコンプ)	文字列の比較
strlen() (ストリングレン)	文字列の長さ

cat は，concatenation (連結) の略

strcpy()

- 文字列は配列なので代入演算子(=)を使って文字配列に代入することはできない．文字列の代入はstrcpy()で行う
  - 文字配列の初期化で「=」記号が出てくるけれども，この「=」は代入演算子ではない！初期化の文法に「=」記号が使われているだけ
- コピー先の文字配列の要素数 >= コピー元の文字列の要素数

文法

```
strcpy(コピー先の文字配列名, コピー元の文字列);
```

例

コピー元の文字列の要素数以上

```
char str1[4], str2[] = "AGU";


strcpy(str1, str2);
puts(str1);
```

実行結果

```
AGU
```

strcat()

- 連結先の文字配列の**最後のヌル文字の位置から**連結元の文字列を連結
  (ヌル文字は連結する文字列の先頭の文字で上書きされる)
- 文字配列の要素数 >= 連結後の文字列の要素数

文法

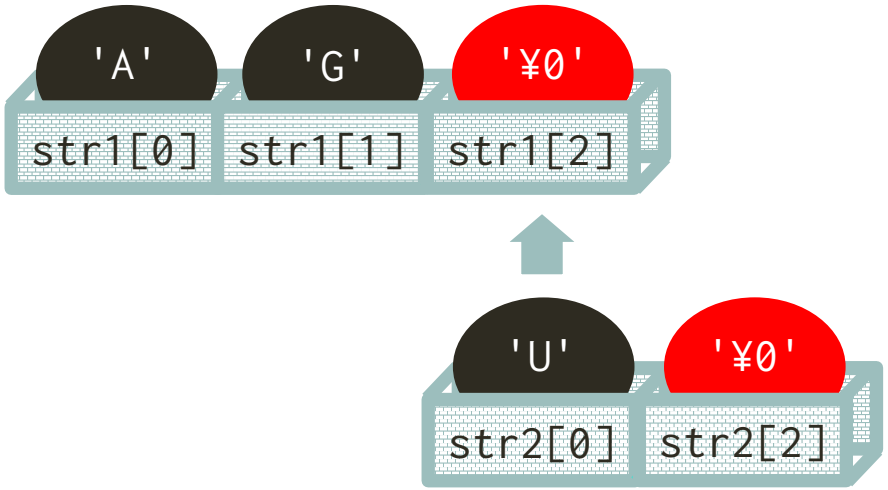
```
strcat(連結先の文字配列名, 連結元の文字列);
```

例

連結後の文字列の要素数

```
char str1[4] = "AG", str2[] = "U";


strcat(str1, str2);
puts(str1);
```



strcmp()

- 2つの文字列が同じならば，戻り値として**0**が返される
  - 基本的にif文の中で用いられる

文法

```
strcmp(文字列1, 文字列2);
```

例

```
char str1[4] = "AGU", str2[] = "AGU";


if(strcmp(str1, str2) == 0) {
   puts("str1とstr2は同じ文字列");
} else {
   puts("str1とstr2は違う文字列");
}
```

実行結果

```
str1とstr2は同じ文字列
```

strlen()

- 戻り値として文字列の長さを返す**(文字列の要素数ではない!)**

文法

```
strlen(文字列);
```

例

```
char str1[] = "AGU", str2[] = "", str3[] = "¥0AGU";


printf("%d¥n", strlen(str1));
printf("%d¥n", strlen(str2));
printf("%d¥n", strlen(str3));
```

実行結果

```
3
0
0
```

課題

レポートの作成

① レポートの冒頭に以下を適切なレイアウトで書く
- 「情報処理実習第12回課題レポート」というタイトル
- 学生番号
- 氏名

② 課題ごとに以下を載せる
- 作成したプログラムのソースコード
- 実行結果を示すコマンドプロンプトの画像

③ 完成したレポートをCoursePower上で提出する