プログラミングI
数理物理，総合理学等向け

2018年11月12日 その1

海谷 治彦

目次

- 前回演習の解答例

- 定数等の定義 – マクロ
- キーボードからの入力とファイル
- 文字列の入力について

- for文

- 今回の演習

復習

配列の基本

- 配列は変数の一種である.
  - 宣言する必要あり.
  - 値を読む(使う)ことができる.
  - 値を書く(更新する)ことができる.
- 通常の変数と異なり，同種の値を複数個扱うことができる.
  - 1個でもいいが，あんまり意味がない.
- 同じ変数名で扱う値群を区別するために，添字という0から始まる番号を用いる.
  - 値を読む場合，書く場合に，変数名に加えて，この添字を指定する必要がある.
- 具体例は，次頁の List 9-2 改を参照.

復習

文字型の配列 文字列

p.239-
(明)

- 単語や文は文字型charの配列で扱うことができる.
- intやfloatの配列と違い，単語や文の終端を表す文字が設定する. (意図的にしないことも可能)

  '¥0'
  - データとしては，全てのビットがゼロのデータ.
- '¥0' の値自体が，while, if 等の条件判断における偽(false)の値となっている.
  - そもそもCでの条件判断は，判断対象の式が，ゼロかそれ以外かを判定しているに過ぎない.
    - ゼロの場合「成り立たない」，それ以外は「成り立つ」と判断.
- 「文字型の配列」のことを単に「文字列」と呼ぶことも多い.

マクロ #define について

p.118-
(明)

- define は，定数的なものを定義する文法である.
- 厳密には定数を定義しているのではなく，コンパイルの前に，ソースプログラムの字面を見て，文字の置き換えを行っているに過ぎない.
  - この前処理を行うプログラムをプリプロセッサと呼ぶ.
- たとえば，

#define MAXLINE 100

- は，以降，ソースプログラム中に現れる MAXLINE という文字を単純に置き換えている.
- ただし，"...." で囲まれる文字列定数内は除く.
- 配列の長さとループの回る回数をそろえる等の場合，便利.
- マクロは慣習として大文字と_のみで書く.

簡単な例

```
// define1.c
#include <stdio.h>

#define MAXARRAY 5

int main(void){
int a[MAXARRAY]={3, 1, 4, 1, 5}, i=0;
  while(i<MAXARRAY){
    printf("%d, ", a[i]);
    i++;
  }
  printf("¥n");
  return 0;
}
```

stdio.h 内で定義済のマクロ

- BUFSIZ
  - 標準的なバッファーのサイズ数.
  - 任意文字列を保管する文字配列のサイズとして使うのもよいだろう.
  - 手元の環境では 512 だったが，環境により異なる.
- NULL
  - 空ポインタ
  - 詳細はポインタの所で話すが，何も指していないことを示す定数.
- EOF
  - ファイル(後述)の終わりをプログラムで識別するための定数.

キーボードからの入力の終わり

- scanf そして教科書[レ]にある gets 等は，キーボードからの入力をプログラム内に取り込む関数群である.
- C言語では，キーボードからの入力を，開始と終了があるテキストファイルとみなしている.
- よって，終了をどう入力するかが重要となる.
- 残念ながらOSによって，これが異なる！
- 標準的なWindows系
  - コントロールを押しながら Z
- UNIX/Linux系
  - コントロールを押しながら D
- この辺は歴史的経緯があるので，統一は難しいだろう.

文字(列)のキーボードからの入力

- 以下の二つの方法を推奨する
- scanf
  - ただし，空白文字を読み込むのが面倒.
    - 空白は文字列の区切りとデフォルトでは判断されるため.
    - そうしないための書式指定はちょっと面倒.
  - 配列名のみ渡し，&はつけない.
- fgets
  - 最も一般的な文字列を読み出す関数.
  - 空白文字に加えて改行文字もデータとして取り込める.
  - 構文が若干複雑. (汎用的にできている)
  - 教科書[レ]にある gets の利用は今は禁止されている.
- それぞれ，一長一短なので，試して欲しいが，基本，この授業では，前者のみ使えばよい.

scanf 定番

```
/*
 scanfで文字列を1つ読み込む定番処理
 細かなエラー処理は省いてある.
 本授業の範囲ではコレで十分. 空白を使うと直前までしか読まれない
 */
#include <stdio.h>
```

```
C:¥LEC¥webpage¥html¥p1¥sample08>cc -o aa scanfstr.c

C:¥LEC¥webpage¥html¥p1¥sample08>aa
test
[test]

C:¥LEC¥webpage¥html¥p1¥sample08>aa
do not use space!
[do]

C:¥LEC¥webpage¥html¥p1¥sample08>
```

```
int main(void){
char str[BUFSIZ]; // 標準バッファーサイズ
  scanf("%s", str); // 文字配列の場合, &はつけない
  printf("[%s]¥n", str); // とりあえず読んだのを表示
  // 以降, 必要な処理を str に対して行なう
  return 0;
}
```

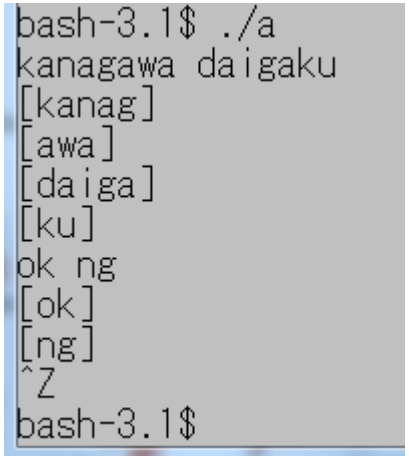
scanfの繰り返し

```
// scanfstr1.c
// 配列数を6としたが，読み込み文字上限は5とすべき
// 長い文字があると途中で切られて，次の入力に入るのがわかる.
// 空白が区切りなのもわかる
// 条件判断時点で str の内容が更新されるのでお勧めしない
#include <stdio.h>

int main(void){
char str[6]; // ¥0分余計にサイズを確保

  while(scanf("%5s",str)!=EOF){
    printf("[%s]¥n", str);
  }

  return 0;
}
```



scanfの繰り返し お勧め版

```
// scanfstr2.c
// 配列数を6としたが，読み込み文字上限は5とすべき
// 長い文字があると途中で切られて，次の入力に入るのがわかる．
// 空白が区切りなのもわかる
// 玄人はこうかかないが，学習目的では妥当．
#include <stdio.h>

int main(void){
char str[6]; // ¥0分余計にサイズを確保
  while(1){
    int result;
    result=scanf("%5s",str);
    if(result==EOF) {break;}
    printf("[%s]¥n", str);
  }

  return 0;
}
```

```
bash-3.1$ ./a
kanagawa daigaku
[kanag]
[awa]
[daiga]
[ku]
ok ng
[ok]
[ng]
^Z
bash-3.1$
```

参考

fgetsの例

```
/* fgets1.c
 空白も改行も
 プログラムに読み込まれているのが分かる */
#include <stdio.h>

#define MAX 5

int main(void){
char str[MAX+1];
// 終端文字 ¥0 が最後に必要なので，MAX+1 文字の宣言が必要
  // 以下は，上限MAX個の文字配列strにキーボードから入力する
  // という意味
  // stdin はキーボードに相当 (正確には標準入力)
  while(fgets(str, MAX+1, stdin)!=NULL){
    printf("[%s]¥n", str);
  }
  return 0;
}
```

```
bash-3.1$ cc fgets1.c
bash-3.1$ ./a
kanagawa daigaku
[kanag]
[awa d]
[aigak]
[u
]
ok ng
[ok ng]
[
]
^Z
bash-3.1$
```

参考

条件判断と値更新の同時記述

- C言語では，if や while 等の条件判断と同時に値更新をするプログラムを記述できる．
- しかも，かなり幅広く用いられている．
  - コンパクトに記述できるため．
- 本授業では，なるだけ，このような同時記述を避けるようにしている．(初学者向け講義のため)

```
while(fgets(str, MAX+1, stdin)!=NULL){
  printf("[%s]¥n", str);
} // fgets1.c より
```

strの更新と，
fgetsの返値のチェックが，
同時に行なわれている．

```
// fgets2.c
while(1){
char* result;
  result=fgets(str, MAX+1, stdin);
  if(result==NULL){break;}
  printf("[%s]¥n", str);
}
```

意図的に，
更新とチェックを分けた．
あまりCでは一般的では
ないが，理解はしやすい．

for文による繰り返し

- C言語には while 以外に for文で繰り返しを書ける.
- 何か機能が違うわけではなく，完全に言い回しの違いのみである.
  - Javaやperl等の後進の言語では機能の違いがある.
- ただ，for文のほうが，N回の処理を繰り返したり，N個の集まり (配列など) を走査したりする記述が，直感的に分かりやすい（かもしれない）.

- 一方，while文は，「処理を繰り返す」ということを，すごく一般的に表現する特徴がある.

whileとforの対比 (一般形)

```
for(カウンタの初期化; 繰返継続条件; カウンタの更新){
    繰り返す処理群;
}
```

同じ意味

```
カウンタの初期化;
while(繰返継続条件){
    繰り返す処理群;
    カウンタの更新
}
```

実際の例 N個の数値を表示

```c
#include <stdio.h>

int main(void){
  int i, n=10;
  for(i=0; i<n; i++){
    printf("%d¥n", i+1);
  }
  return 0;
} // nfor.c
```

```c
#include <stdio.h>

int main(void){
  int i, n=10;
  i=0;
  while(i<n){
    printf("%d¥n", i+1);
    i++;
  }
  return 0;
} // nwhile.c
```

```
C:¥LEC¥P1¥2014¥p109sample>a.exe
1
2
3
4
5
6
7
8
9
10
```

for文の制約

- カウンタの更新は，繰り返す処理が終わった後にfor文では行なわれる．
- よって，while文のように，カウンタの更新を繰り返し文の中の自由な位置で行なうことはできない．

この位置でのカウンタの更新は，for文にうまくはまらない．

```
#include <stdio.h>

int main(void){
  int i, n=10;
  i=0;
  while(i<n){
    i++;
    printf(“%d¥n”, i);
  }
  return 0;
} // nwhile2.c
```

カウンタ更新部分が空欄となる．

```
#include <stdio.h>

int main(void){
  int i, n=10;
  for(i=0; i<n;){
    i++;
    printf("%d¥n", i);
  }
  return 0;
} // nfor2.c
```

参考: N個数えじゃないfor文

- for文はN回繰り返しの利用が圧倒的に多い.
- しかし，文法的には，自由に書ける．以下例.

```
#include <stdio.h>

int main(void){// ユークリッド互除法による最大公約数の計算
  int a, b;
  for(a=8, b=12; a!=b; b -=a){ // 複数の初期値が設定可
    if(a>b){ // bのほうが常に大きい数が来るよう値の入れ替え
    int tmp;
      tmp=a; a=b; b=tmp;
    }
  }
  printf("gcd=%d¥n", a);
  return 0;
} // gcd.c
```

次のスライドに