ネットワークプログラミング演習（第 4 週〜第 6 週）

——C 言語の基礎（その 2）——

真部　雄介，八島　由幸，中川　泰宏

5 月 10 日，5 月 17 日，5 月 24 日

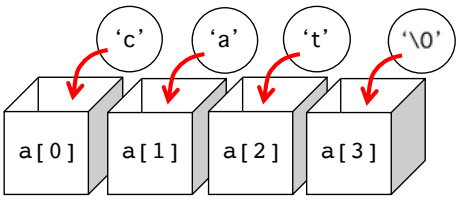
1　C 言語における文字列の扱い方

1.1　文字列と配列

- 文字列（String）：文章や単語などの複数の文字からなるデータのこと．
- C 言語に文字列型は用意されていない
- 文字列は char 型の配列として扱い，文字配列と呼ぶ
- １つの文字はシングルクォーテーション記号で囲み，文字列（複数の文字）はダブルクォーテーション記号で囲む
- 文字列を格納する例
  - ダブルクォーテーションを使った格納の例を示す．

```
char a[] = "cat";
```

  - これは，文字配列 a[] に以下のようにデータを格納したことを意味する．



  - また，以下のようにシングルクォーテーションを使って，文字配列 a[] に文字を１つ１つ格納することと同じことを意味する．

```
char a[] = { 'c', 'a', 't', '\0' };
```

  - '\0'の意味＝文字列終わりを表す．ヌル文字と呼ばれる．Windows 環境では'¥0'と表すので注意．
- 一般的に，文字配列の先頭の要素から順に検索していき，'\0'が見つかったところまでをひとかたまりの文字列とみなす
- 必ず最後に'\0'が必要なため，配列の要素数は文字数＋１以上必要であることに注意
- 講義では，**半角英数字のみ**（1 バイト文字のみ）扱い，日本語（2 バイト，3 バイト文字）は扱わないこととする．

1.2　文字列の標準入出力

- 以下は，文字列の標準入出力方法を示したプログラム「string_io.c」である．

```
01:/* string_io.c */
02:#include <stdio.h>
03:
04:int main(void){
05:    char str[128]; /* 読み込む文字列を格納する文字配列 */
06:    int i;  /* ループ用の制御変数 */
07:
08:    /* 標準入力 */
09:    printf("Input a string, please.\n");
10:    scanf("%s",str);
11:
12:    /* 標準出力1 */
13:    printf("%s\n",str);
14:
15:    /* 標準出力2 */
16:    i = 0;
17:    while( str[i] != '\0' ){ /* 格納された文字が'\0'でない間，繰り返す */
18:       printf("%c",str[i]); /* 先頭から順に1文字ずつ出力 */
19:       i++;
20:    }
21:    printf("\n");
22:
23:    return 0;
24:}
```

  - 5 行目のchar str[128]は，127 文字（'\0'のぶん-1）まで入力できるという意味．十分な大きさの配列を用意すること．
  - 10 行目と 13 行目：文字列に対する入出力変換指定子は %s である．
  - 変換指定子%sと対応付けられる変数の指定がstrとなっている．
  - 配列変数は，添え字を付けずに名前だけを指定すると，配列の先頭のアドレス を示す．
  - 標準出力 1（13 行目）は，もっとも一般的な出力方法．
  - 標準出力 2（16〜21 行目）は，1 文字ずつ文字配列を参照して出力する方法．

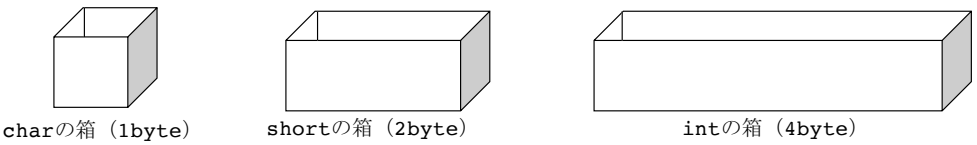
- 実行結果を以下に示す．

```
Input a string, please.
Hello!
Hello!
Hello!
```

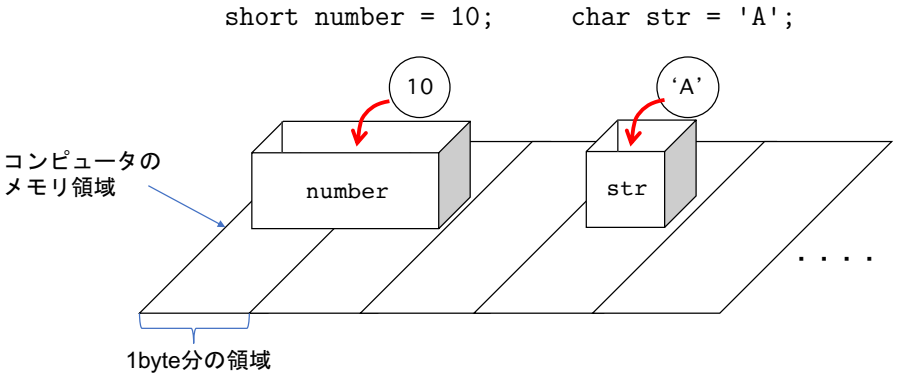
  - 1 番目の「Hello!」＝標準入力（10 行目のscanfに渡されるデータ．自分で入力する行）
  - 2 番目の「Hello!」＝出力 1（13 行目の出力）
  - 3 番目の「Hello!」＝出力 2（16〜21 行目の出力）

2　アドレスとポインタ変数

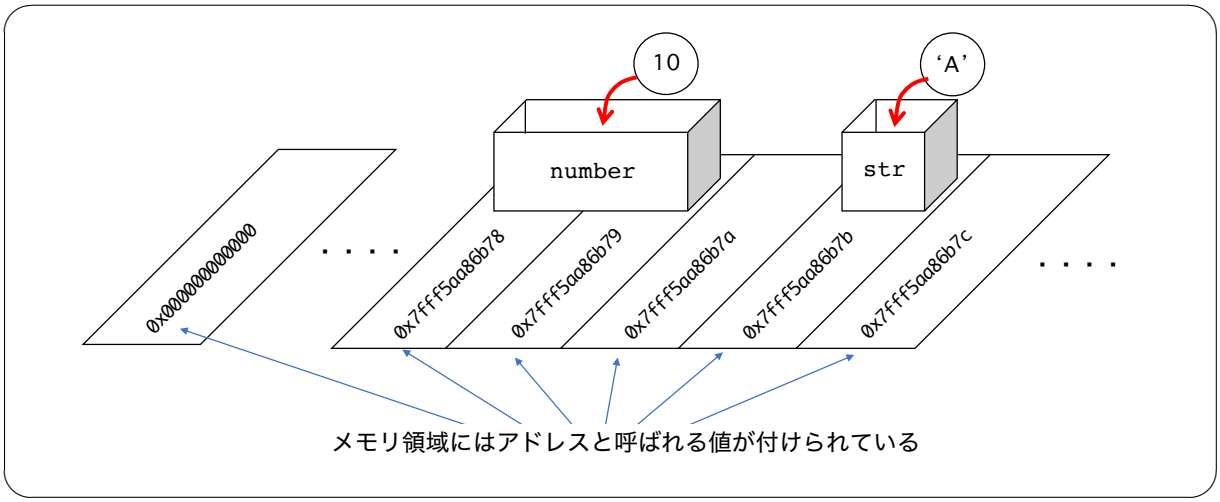
- 変数の宣言は，データ型の **サイズ** に応じた箱を用意するイメージに近い．



- プログラムが実行されるとき，コンピュータ内部では，全データが **メモリ** 上に展開される
- 変数を宣言する＝箱を用意する＝ **メモリ領域** を確保する



- コンピュータのメモリ領域は，1byte ごとに番号が付けられて管理されている．メモリ領域につけられた番号，すなわち，メモリ領域の場所を示す値は **アドレス** と呼ばれる．



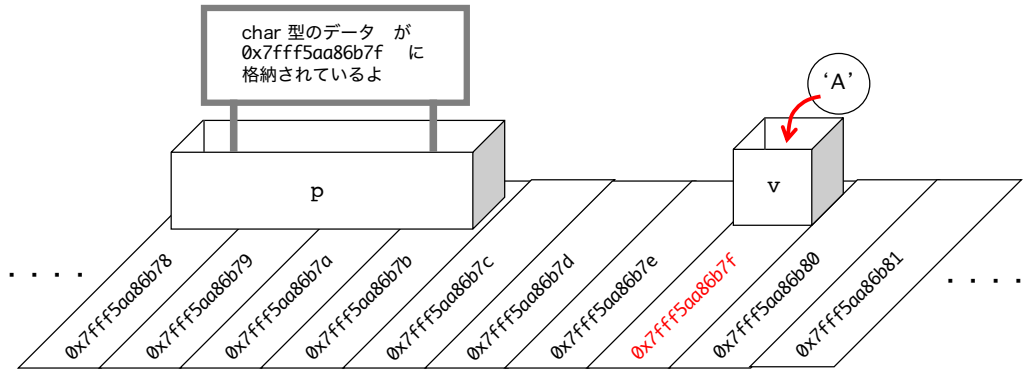
- C 言語では，アドレスを利用してプログラムを記述する必要がある場合がある．データが格納されている**アドレスを格納するための変数**を ポインタ変数 といい，変数名の前に “*” 記号をつけて宣言する[*1]．

```
int *a;
char *b;
```

ここで，“*” 記号は， 間接参照 演算子と呼ばれる．

- ポインタ変数は,整数,実数,文字といった一般的なデータを 代入することはできない
- ポインタ変数に代入できるのは， アドレス のみである．変数のアドレスは，変数名の前に “&” 記号をつけて表す．

```
char v = 'A';    /* char型の変数vを宣言し，'A'を代入 */
char *p;         /* char型の変数のアドレスを代入するポインタ変数pを宣言 */
p = &v;          /* vのアドレスをpに代入 */
```



ここで，“&” 記号は， アドレス 演算子と呼ばれる．

[*1] ポインタ変数を宣言すると，どんなデータ型でも固定長のメモリ領域が確保される．確保されるサイズは処理系に依存するが，ここでは 4bytes 分のメモリ領域が確保されるということにして図示している．

- ポインタ変数を用いたプログラム「pointer.c」を示す.

```
01:/* pointer.c */
02:#include <stdio.h>
03:
04:int main(void) {
05:    int val;  /* int型の値を代入する変数 */
06:    int *ptr; /* int型変数のアドレスを代入するポインタ変数 */
07:
08:    val = 10; /* int型の値10を代入 */
09:    ptr = &val; /* 変数valのアドレスを代入．int型の値は代入できない */
10:
11:    /* valのアドレス，valの値を出力 */
12:    printf("=== val ===\n");
13:    printf("address:%p, value:%d\n",&val, val);
14:    /* ptrの値，ptrの示す場所に格納された値を出力 */
15:    printf("=== ptr ===\n");
16:    printf("address:%p, value:%d\n",ptr, *ptr);
17:
18:    *ptr = 20; /* ptrの示す場所に格納された値を20に変更 */
19:
20:    /* valのアドレス，valの値を出力 */
21:    printf("=== val ===\n");
22:    printf("address:%p, value:%d\n",&val, val);
23:    /* ptrの値，ptrの示す場所に格納された変数の値を出力 */
24:    printf("=== ptr ===\n");
25:    printf("address:%p, value:%d\n",ptr, *ptr);
26:
27:    return 0;
28:}
```

  - 9 行目：int型変数valの**アドレス**をポインタ変数ptrに代入.
  - 18 行目：*ptrは,「ポインタ変数ptrの指し示す**アドレスの場所にある変数**」を表す．すなわち，ここではvalを表す.
  - 「ptr」にはアドレスしか代入できない.
  - 「*ptr」とすると，アドレスの指し示す場所にある変数を表すので，値の代入や参照ができる.
  - %p はアドレスを 16 進数で表示する出力変換指定子

- 実行結果は以下のようになる．アドレスの値は環境によって異なるので注意．

```
=== val ===
address:0018FF54, value:10
=== ptr ===
address:0018FF54, value:10
=== val ===
address:0018FF54, value:20
=== ptr ===
address:0018FF54, value:20
```

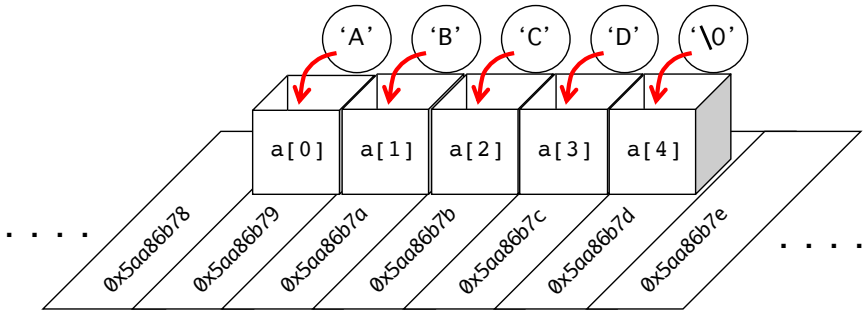
  - プログラムの 18 行目で，*ptr = 20と記述しただけで，valの値が変化している．
  - ポインタ変数ptrを用いて，変数valへの値の代入ができている＝ 間接参照

3　文字列とポインタ変数：ポインタ変数の応用 1

- 次のように配列a[]を宣言した場合，その要素分のメモリ領域が確保される．

```
char a[] = "ABCD"; または
char a[] = {'A', 'B', 'C', 'D', '\0'};
```

- このとき，メモリ領域は以下のようになっている（アドレスは適当です）．



- 文字配列は，メモリ領域に 順番に 確保される．
- このことを利用すると，文字列は配列としてだけでなく， ポインタ変数 を使って操作することもできる．

- 配列表現とポインタ表現は以下のような対応がある.

値	'A'	'B'	'C'	'D'	'\0'
アドレス	0x542f3b78	0x542f3b79	0x542f3b7a	0x542f3b7b	0x542f3b7c
配列表現	a[0]	a[1]	a[2]	a[3]	a[4]
ポインタ表現	*a	*(a+1)	*(a+2)	*(a+3)	*(a+4)

- 配列変数とポインタ変数を用いた文字列操作プログラム「array_pointer.c」を以下に示す.

```
01:/* array_pointer.c */
02:#include <stdio.h>
03:
04:int main(void){
05:    char a[] = "ABCD";
06:    char *b = "EFGH";
07:    int i;
08:
09:    /* 配列変数をポインタを用いて1文字ずつ出力 */
10:    for( i = 0; i < 4 ; i++){
11:        printf("%c", *(a+i));
12:    }
13:    printf("\n");
14:
15:     /* ポインタ変数を配列を用いて1文字ずつ出力 */
16:    for( i = 0; i < 4 ; i++){
17:         printf("%c", b[i]);
18:    }
19:    printf("\n");
20:
21:   return 0;
22:}
```

  - 10 行目〜12 行目：配列変数aをポインタを用いて操作
  - 16 行目〜18 行目：ポインタ変数bを配列を用いて操作
  - つまり，配列変数として宣言してポインタ変数として操作したり，ポインタ変数として宣言して配列変数として操作することができる.

文字列に関する配列変数とポインタ変数のまとめ

配列変数とポインタ変数の対応をしっかり身につけよう．

	配列変数	ポインタ変数
宣言	`char a[5];`	`char *a`
初期化	`char a[] = "ABCD";`	`char *a = "ABCD";`
代入	`a[0] = 'A'; a[1] = 'B'; ...`	`a = "ABCD";`
データサイズ	固定（宣言時に決める）	初期化，代入されるまで不定
各要素の参照	`a[i]`	`*(a+i)`
文字列の先頭アドレス	`a`	`a`

4　動的配列としてのポインタ変数：ポインタ変数の応用 2

- 第 1 回の講義で，10 個の整数の中から最小値を求めるプログラム「`minimum.c`」を学んだ．

```
00:/* minimum.c */
01:#include <stdio.h>
02:
03:int main(void) {
04:     int data[] = {6,3,7,2,1,5,7,3,5,9};
05:     int mini;
06:     int i;
07:
```

```
08:     mini = data[0];
09:     for( i = 1; i < 10; i++ ){
10:         if( mini > data[i] ){
11:             mini = data[i];
12:         }
13:     }
14:
15:     printf("The min. is %d.\n", mini);
16:
17:     return 0;
18:}
```

- このプログラムは，最小値を求める整数の個数が **固定** だった．
- 「任意の個数」の整数の中から最小値を求めるプログラムにするにはどうすればよいか？

4.1　解決法

- どのようなデータ型の値を確保する必要があるかはわかっていても，そのサイズまで事前に確定できない場合，ポインタ変数を宣言しておき，あとから必要な分のメモリ領域（配列の数）を動的に確保することができる．

```
01:/* minimum3.c */
02:#include <stdio.h>
03:#include <stdlib.h>
04:
05:int main(void) {
06:     int *data;    /* ポインタ変数として宣言 */
07:     int n;        /* 入力する整数の個数 */
08:     int minimum;  /* 最小値を格納するための変数 */
09:     int i;        /* 繰り返し用の制御変数 */
10:
11:     /* 整数の個数を入力 */
12:     printf("How many? :");
13:     scanf("%d",&n);
14:
15:     /* int型の変数をn個格納するためのメモリ領域を確保 */
16:     data = (int *)malloc(sizeof(int)*n);
17:
18:     /* n回の整数の入力を繰り返す */
19:     for( i = 0; i < n ; i++ ){
20:         scanf("%d",&data[i]);
21:     }
22:
23:     minimum = data[0]; /* 配列の先頭の値をとりあえず最小値とする */
24:     for( i = 1; i < n; i++ ){   /* 配列の2番目の値から順番に見ていく */
25:         if( minimum > data[i] ){ /* 最小値より小さな整数が見つかったら */
26:             minimum = data[i];   /* その値を最小値として更新する */
27:         }
28:     }
29:
30:     printf("The minimum number is %d.\n", minimum);
31:
32:     free(data);  /* メモリ領域の開放 */
33:
34:     return 0;
35:}
```

- 6 行目：ポインタ変数dataを定義
- 13 行目：読み込むデータの個数nを入力する
- 16 行目：malloc関数を呼び出して，データの個数n個分のメモリ境域を確保する．malloc関数の書式を以下に示す．

```
ポインタ変数名 = (データ型 *)malloc( データサイズ )
```

  minimum3.cでは，int型変数 1 つ分のデータサイズをsizeof演算子で算出し，それをデータの個数n個分確保している．
- 19〜21 行目：確保した領域に先頭から順番に配列表現を用いて整数を入力する．
- 32 行目：free関数を使って，確保したメモリ領域を開放する．
- 3 行目でstdlib.hをインクルードしていることに注意．このインクルードのおかげで，malloc関数やfree関数が使える．

- 実行結果

```
How many? :5
9
5
3
8
4
The minimum number is 3.
```

5　ファイル入出力

5.1　ストリーム

- **ストリーム**：バッファを持つ情報の論理的な流れ道のこと．
  →C 言語では **FILE 型のポインタ** を用いる．
- ファイル入出力の手順：ファイルオープン → 読み・書きの処理 → ファイルクローズ
  ・ファイルオープン：ストリームとファイルを **つなぐ** こと
  ・ファイルのクローズ：ストリームからファイルを **切り離す** こと
- ファイルのオープンに用いる関数（fopen）

書式	引数	戻り値
`FILE *fopen(char *file, char *mode)`	第一：ファイル名 第二：モード	ファイルポインタ

  – fileが示すファイルをmodeが示すモードでオープンする
  – 読み込みモード（"r"）：ストリームを通してファイルを読み込める状態にする．第一引数に指定したファイルが存在しない場合は NULL を返す．
  – 上書きモード（"w"）：ストリームを通してファイルに書き込める状態にする．第一引数に指定したファイルが存在しない場合は，**新しくファイルを作成する**．指定したファイルが既に存在する場合は**元の内容が上書きされる**．書き込み許可のないファイルを指定した場合は NULL を返す．
  – 追加モード（"a"）：ストリームを通してファイルに書き込める状態にする．第一引数に指定したファイルが存在しない場合は，**新しくファイルを作成する**．指定したファイルが既に存在する場合は**元の内容の終わりから書き足される**．書き込み許可のないファイルを指定した場合は NULL を返す．
  – FILE構造体：stdio.hに定義されている入出力ファイルの情報を扱うために定義された構造体[*2]．
- ファイルのクローズに用いる関数（fclose）

書式	引数	戻り値
`int fclose(FILE *fp)`	ファイルポインタ	ファイルをクローズした結果

  – fpが示すファイルをクローズする
  – 切り離し（クローズ）に成功した場合は 0 を返す．失敗した場合はEOF[*3]を返す．
- ファイルオープン・クローズの基本パターン

[*2] 構造体については第 7 回で扱う予定です．
[*3] EOF は-1 として定義されている定数．ファイル終わり（End of File）を表す値として用いられる．

```
#include <stdio.h>

int main(){
   FILE *fp; /* ファイルポインタの宣言 */

   /* test.txtというファイルを読み込みモードでオープン */
   fp = fopen("test.txt", "r");
   if( fp == NULL ){  /* オープンできなかたっときのエラー処理 */
      fprintf(stderr, "Cannot open file\n");
      return -1;
   }

   /* ---- ファイルの読み込みや書き込み処理を書く部分 ---- */

   fclose(fp); /* ファイルをクローズするのを忘れずに */

   return 0;
}
```

ファイルへのポインタが NULL のときは，実行時エラー（セグメント例外）が発生するため，**エラー処理** を記述するのが原則.

5.2 バイト入出力関数

- バイト入出力関数とは，**1 バイト** 単位 (1 文字単位) で入出力を行う関数である.
- ファイルに対して読み書きを行うものと，標準入出力に対して読み書きを行うものがある.
- ファイルに対するバイト入力関数（fgetc，getc）

書式	引数	戻り値
int fgetc(FILE *fp) int getc(FILE *fp)	ファイルポインタ	読み込んだ文字

  - fpが示す FILE 構造体で指定されているファイル位置指示子が示す文字を 1 文字読み込み，ファイル位置指示子の示す位置を 1 つすすめる.
  - 戻り値：成功の場合は読み込んだ文字（int 型），失敗はの場合はEOFを返す
- ファイルに対するバイト出力関数（fputc，putc）

書式	引数	戻り値
int fputc(int c, FILE *fp) int putc(int c, FILE *fp)	第一：読み込む文字の格納用変数 第二：ファイルポインタ	書き込んだ文字

- fpが示す FILE 構造体で指定されているファイル位置指示子が示す位置へ変数cを書き出し，ファイル位置指示子の示す位置を 1 つすすめる．
- 戻り値：成功の場合は書き出した文字（int 型），失敗の場合はEOFを返す

- 標準入出力に対する入出力関数（getchar，putchar）

書式	引数	戻り値
int getchar()	なし	読み込んだ文字
int putchar(int c)	書き出す文字	書き出す文字

- getchar：標準入力から 1 文字読み込む．戻り値は，成功の場合は入力文字（int 型），失敗の場合はEOFを返す
- putchar：標準出力に 1 文字（c）を書き出す．成功の場合は出力文字（int 型），失敗の場合はEOFを返す

- プログラム例と実行結果

  この例では，getchar関数とputc関数を使用している．

```
01:/* byte_io.c */
02:#include <stdio.h>
03:
04:int main(void){
05:   FILE *fp;
06:   int c;
07:
08:   fp = fopen("mail.txt","w");
09:   if( fp == NULL ){
10:      fprintf(stderr, "Cannot open file.\n");
11:      return -1;
12:   }
13:
```

. . . . . 次のページへ続く . . . . .

```
14:   while( (c = getchar()) != EOF ){ /* 標準入力から一文字入力を繰り返す */
15:      putc(c, fp); /* ファイル「mail.txt」に書き出し */
16:   }
17:
18:   fclose(fp);
19:
20:   return 0;
21:}
```

– このプログラムは，EOFが入力されるまで，標準入力から 1 文字入力を繰り返す．
– したがって，スペースや改行も入力できる．
– Windows のコマンドプロンプト上で，標準入力からEOFを入力するには，「Ctrl+Z」を入力し，[enter] キーを押す．
– (Linux や Mac のコンソールを使っている人は「Ctrl+D」を入力)

実行結果は以下のようになる．

```
Taro Koudai: taro@a.chibakoudai.jp
Hanako Chiba: hana@a.chibakoudai.jp
^Z
続行するには何かキーを押してください．．．
```

プログラム終了後, 入力した情報が書き込まれた「mail.txt」というファイルが出力される. EasyIDEC を使っている人は，実行したプログラムの project フォルダの中にファイルが出力されているはずである．

5.3 文字列入出力関数

- NULL 文字（'\0'）で終わる文字列データの入出力を行う関数.
- 文字列入力関数

書式	引数	戻り値
char *fgets (char *b, int n, FILE *fp)	第一：読み込んだ文字列の格納用 第二：読み込む最大文字数 第三：ファイルポインタ	入力文字列

– fpが示す FILE 構造体で指定されているファイル位置指示子が示す文字列をbに読み込む.
– 文字列は，「n 文字まで」,「改行まで」,「ファイルの終わりまで」のいずれかの条件が満たされるまで読み込まれる．
– 戻り値：成功の場合＝文字配列へのポインタ，失敗の場合＝ NULL

- 文字列出力関数

書式	引数	戻り値
int fputs(char *b, FILE *fp)	第一：書き出す文字列 第二：ファイルポインタ	書き出した結果

  - fpが示す FILE 構造体で指定されているファイル位置指示子が示す位置へ文字列bを書き出す.
  - 文字列終わりを示す NULL 文字は書き換える.
  - 戻り値：成功の場合＝非負の整数，失敗の場合＝EOF

- プログラム例と実行結果

```
01:/* line_io.c */
02:#include <stdio.h>
03:
04:#define MAXLEN 64 /* 文字列の最大長 */
05:
06:int main(void){
07:   FILE *fp1, *fp2;
08:   char data[MAXLEN];
09:
10:   /* 読み込むファイルをオープン */
11:   if( (fp1=fopen("mail.txt","r")) == NULL ){
12:      printf("Cannot open file: mail.txt\n");
13:      return -1;
14:   }
15:   /* 書き出すファイルをオープン */
16:   if( (fp2=fopen("backup.txt","w")) == NULL ){
17:      printf("Cannot open file: backup.txt\n");
18:      return -1;
19:   }
20:
21:   /* fp1が示すファイルから一行ずつ読み込みを繰り返す */
22:   while( fgets(data, MAXLEN, fp1) != NULL ){
23:      fputs(data, fp2); /* 読み込んだ一行をfp2が示すファイルに書き出す */
24:   }
25:
```

. . . . . 次のページへ続く . . . . .

```
26:    fclose(fp1);
27:    fclose(fp2);
28:
29:    return 0;
30:}
```

5.4　フォーマット化入出力関数

- 文字，文字列，数値からなるさまざまな形式のデータを，全て文字列として扱うための入出力関数.
- ファイルからのフォーマット化入力関数（fscanf）

書式	引数	戻り値
int fscanf (FILE *fp, char *format, ...)	第一：ファイルポインタ 第二以降：書式文字列と 変数（scanf 関数と同じ）	入力項目数

  - fpが示す FILE 構造体で指定されているファイルからformat記述で成型した文字列または数値を読み込む
  - 第一引数に stdin を指定すると標準入力となり，scanf関数と同じになる
  - 戻り値：成功の場合＝入力項目数（int），失敗の場合＝EOF
- ファイルへのフォーマット化出力関数（fprintf）

書式	引数	戻り値
int fprintf (FILE *fp, char *format, ...)	第一：ファイルポインタ 第二以降：書式文字列と 変数（printf 関数と同じ）	出力バイト数

  - format記述で成型した文字列または数値を，fpが示す FILE 構造体で指定されているファイルへ書き出す
  - 第一引数に stdout を指定すると標準出力となり，printf関数と同じになる
  - 戻り値：成功の場合＝出力バイト数（int），失敗の場合＝負の値（int）
- 文字列からのフォーマット化入力関数（sscanf）

書式	引数	戻り値
int sscanf (char *b, char *format, ...)	第一：文字配列へのポインタ 第二以降：書式文字列と 変数（scanf 関数と同じ）	入力項目数

  - bが示す文字列からformat記述で成型した文字列または数値を読み込む

  - 戻り値：成功の場合＝入力項目数（int），失敗の場合＝EOF

- 文字列へのフォーマット化出力関数（sprintf）

書式	引数	戻り値
int sprintf （char *b, char *format, ...）	第一：文字配列へのポインタ 第二以降：書式文字列と 変数（printf 関数と同じ）	出力バイト数

  - format記述で成型した文字列または数値を，bが指し示す文字列へ書き出す
  - 戻り値：成功の場合＝出力バイト数（int），失敗の場合＝負の値（int）

6　文字列操作関数群の利用

文字列操作関数群を使うには，string.h というヘッダファイルをインクルードする．

6.1　文字列をコピーする関数 strcpy

書式	引数	戻り値
char *strcpy(char *s1, char *s2)	２つの文字列	コピー先の文字列

- コピー元文字列（s2）の NULL 文字を含めて，コピー先文字列（s1）が示すポインタの位置にコピーする
- 戻り値：コピー先文字列（s1）へのポインタ（s1にs2をコピーするのでs1の先頭のアドレス）

6.2　文字列を連結する関数 strcat

書式	引数	戻り値
char *strcat(char *s1, char *s2)	２つの文字列	文字列を連結した結果

- s1にs2を連結し，NULL 文字を付加する．s1の NULL 文字は上書きされる．
- 戻り値：連結された文字列へのポインタ（s1の後ろにs2を連結するのでs1の先頭のアドレス）

6.3　文字列を比較する関数 strcmp

書式	引数	戻り値
int strcmp(char *s1, char *s2)	２つの文字列	文字列を比較した結果

- s1とs2の文字コードを先頭アドレスから順次 1 バイト（1 文字）ずつ比較する．NULL 文字または文字列が同じではないことがわかるまで比較する．
- 比較していく際，等しく無い文字が現れた場合にその 1 バイトの大小で全体の大小を判定する．
- 文字列の長さの大小ではないので注意．

- 戻り値：int型の値．s1==s2のとき__0__，s1<s2のとき__-1__，s1>s2のとき__1__．

6.4 文字列の長さを返す関数 strlen

書式	引数	戻り値
int strlen(char *str)	文字列	引数str の長さ（バイト単位）

6.5 文字列操作関数の使用例

```
01:/* str_example.c */
02:#include <stdio.h>
03:#include <string.h>
04:
05:#define MAXLEN 64 /* 文字列の最大長 */
06:
07:int main(void){
08:     char s1[MAXLEN] = "Chiba ",  s2[MAXLEN] = "";
09:     char s3[MAXLEN] = "University";
10:     char s4[MAXLEN] = "Institute of Technology";
11:     int result;
12:
13:     /* コピーと連結 */
14:     strcpy(s2, s1);
15:     strcat(s1, s3);
16:     strcat(s2, s4);
17:     printf("s1:%s\ns2:%s\n\n",s1, s2);
18:
19:     /* 比較 */
20:     result = strcmp(s1, s2);
21:     if( result > 0 ){  printf("%s > %s\n", s1, s2); }
22:     else if( result < 0 ){ printf("%s < %s\n", s1, s2); }
23:     else{ printf("%s = %s\n", s1, s2); }
24:
25:     return 0;
26:}
```

7　キャスト変換とデータ型変換関数

7.1　キャスト変換

- C 言語では，演算結果のデータ型が一致していない場合は，データ型の **統一** を行う．

優先順位	データ型
1	long double
2	double
3	float
4	unsigned long int
5	long int
6	unsigned int
7	int
8	unsigned short int, unsigned char
9	short int, char

- データ型には優先順位がある．例えば，int型の値とfloat型の値を用いて計算する場合，float型の優先順位の方が高いので，int型の値をfloat型の値に変換して演算を行う．
- 算術演算でデータ型変換が暗黙的に行われる例と行われない例を示す．

```
01:/* change.c */
02:#include <stdio.h>
03:
04:int main(void){
05:    float a, b, c;
06:
07:    a = (1 + 2 + 3) / 4;
08:    b = (1.0 + 2 + 3) / 4;
09:    c = 1 / 2 * 3.0;
10:    printf("a=%f, b=%f, c=%f\n", a, b, c);
11:
12:    return 0;
13:}
```

実行結果は以下のようになる．

```
a=1.000000, b=1.500000, c=0.000000
```

  - 7 行目：(1+2+3)の結果＝ int 型 ，(1+2+3)/4の結果＝ int 型 ，
    変数aへの代入＝暗黙的な型変換で float 型
  - 8 行目：(1.0+2+3)の結果＝ float 型 ，(1.0+2+3)/4の結果＝ float 型 ，
    変数bへの代入＝型変換無しで float 型
  - 9 行目：1/2の結果＝ int 型 ，1/2*3.0の結果＝ float 型 ，
    変数cへの代入＝型変換無しで float 型
- 強制的にデータ型の変換を行う方法を キャスト変換 という．
- change.cの演算結果が正しくなるようにキャスト変換すると以下のようになる

```
07:     a = (float)(1 + 2 + 3) / 4;
08:     b = (1.0 + 2 + 3) / 4;
09:     c = (float)1 / 2 * 3.0;
```

  - (float)の部分を キャスト演算子 という．変換したいデータ型を括弧で囲んで変換したい値の前に記述する．
  - 7 行目：(1+2+3)の結果を(float)にキャスト変換することで，小数点以下の計算結果を扱えるようにしている
  - 9 行目：1を(float)にキャスト変換することで，1/2の結果がfloatとなるので，小数点以下の計算結果を扱えるようにしている

実行結果は以下のようになる．

```
a=1.500000, b=1.500000, c=1.500000
```

7.2　データ型変換関数

- stdlib.h には，以下のようなデータ型変換関数が用意されている．

書式	引数	戻り値
int atoi(char *str)	文字列	引数を int 型 に変換した値
double atof(char *str)	文字列	引数を double 型 に変換した値

- いずれの関数も，引数に与えられた文字列が変換不可の場合は 0 を返す
- ファイルからの数値データ読み込み処理の際によく利用される．
  → fscanf 関数を使って 文字列 としてデータを読み込み，これらの関数を使って所望のデータ型に変換する

- データ型変換関数のプログラム例と実行結果

```
01:/* atosome.c */
02:#include <stdio.h>
03:#include <stdlib.h>
04:
05:int main(void){
06:    char A[] = "12.3";
07:    int B;
08:    double C;
09:
10:    B = atoi(A); /* int型に変換（小数点以下は無視される） */
11:    C = atof(A); /* double型に変換 */
12:
13:    printf("BxB=%d, CxC=%lf\n", B*B, C*C);
14:
15:    return 0;
16:}
```

実行結果は以下のようになる．

```
BxB=144, CxC=151.290000
```

8　乱数，数学ライブラリ

8.1　乱数

- 意味や規則性のまったくない数字の列を乱数という．
- 乱数に関する関数群は，__stdlib.h__ に定義されている．
- コンピュータプログラムは算術演算で数字列を発生させるため，真の乱数を発生させることは本質的に困難である．→ プログラムで発生させることができる乱数は，__擬似乱数__ と呼ばれる．
- 算術演算の初期状態が同じなら，発生する乱数は__同じ__ものになる．
- 算術演算の初期状態は，__srand__ 関数で設定することができる．
- 乱数を発生させる関数は__rand__ 関数で，引数は取らず，0〜RAND_MAXまでの疑似乱数を返す関数である．

- 乱数を発生させるプログラム例と実行結果

```
01:/* random.c */
02:#include <stdio.h>
03:#include <stdlib.h>
04:
05:#define MAX 100
06:
07:int main(void){
08:   int i = 0;
09:   int n;
10:   unsigned int seed;
11:
12:   printf("Seed? = ");
13:   scanf("%d", &seed); /* 乱数のシード（種）を入力 */
14:
15:   srand(seed); /* シードを設定し乱数を初期化 */
16:
17:   while(i < 10){
18:      /* 1〜100まで整数をランダムに発生させる */
19:      n = rand() % MAX + 1;
20:      printf("%d ",n);
21:      i++;
22:   }
23:   printf("\n");
24:
25:   return 0;
26:}
```

実行結果は以下のようになる．

```
Seed? = 8967
8 50 52 72 33 41 85 38 26 40
```

8.2 数学ライブラリ（math.h）

- C 言語には高度な数値計算を行うための数学ライブラリ（math.h）が用意されている
- 本節で示す関数は，math.h をインクルードする必要がある．

- Linux や Mac の環境を使っている人は，コンパイラにmathライブラリがリンクされていない場合は，コンパイル時に libm.a を明示的にリンクすることにより使用可能となる．具体的には，コンパイル時に以下のように「-lm」を付加する（やらなくてもコンパイルが通るならやらなくてもよい）．

```
$ gcc -lm math_test.c
$ ./a.out
```

- 代表的な関数を以下に示す．

書式	引数	戻り値
double sin(double x)	角度（ラジアン）	引数の正弦（$\sin x$）
double cos(double x)	角度（ラジアン）	引数の余弦（$\cos x$）
double tan(double x)	角度（ラジアン）	引数の正接（$\tan x$）
double log(double x)	数値	引数の自然対数（$\log x$）
double log10(double x)	数値	引数の常用対数（$\log_{10} x$）
double exp(double x)	数値	引数の指数関数（e^x）
double pow(double x, double y)	数値	引数の累乗（x^y）
double sqrt(double x)	数値	引数の平方根（$\sqrt{x}$）
double fabs(double x)	数値	引数の絶対値（$\|x\|$）

- プログラム例と実行結果

```
01:/* math_test.c */
02:#include <stdio.h>
03:#include <math.h>
04:
05:int main(void){
06:    double X = 2.0;
07:
08:    printf("%f %f\n", sqrt(X), pow(X, 0.5));
09:    printf("%f %f\n", pow(X, 2), X*X);
10:
11:    return 0;
12:}
```

実行結果は以下のようになる．

```
1.414214 1.414214
4.000000 4.000000
```