● 今日の実習

【サンプルプログラム】

★ ex14-1-1.c **標準入力からのテキストデータの読み込み**.

1. このプログラムは「標準入力」からテキストデータを読み, それを標準出力に書き出すだけのものである.
2. “feof(stdin)” とは標準入力 (stdin) が入力終了となるときに 1 を返し, そうでない状態の時には 0 を返す関数である.
3. 標準入力から１行ごとまたは最大 MAX_CHAR_PAR_LINE 文字を一度に buf に読み込もうとしている.
4. 実際に読み込みを行うのは fgets 関数であり, fgets は「データを格納する」 buf, 「最大読み込みデータ量」 MAX_CHAR_PAR_LINE, 「読み込み先」 stdin を渡している.
   buf には「改行文字」を含めて読み込まれ, 末尾には NULL 文字が付加される.
5. fgets は現在の「ファイルポインタ」（現在のファイル上の読み込み位置）の位置から読み込みを始め, fgets は読み込みが終わると, ファイルポインタを読み込んだ末尾の文字の次に移動する. したがって, 最後の行の末尾まで読み込むと, ファイルポインタの位置は EOF（ファイル終端）の直前に到達する.
   fgets の返却値は, 読み込んだ文字列へのポインタである. ファイルポインタが EOF にあるときに fgets を実行すると NULL が返却され, buf の中身は変更されない.
6. このプログラムを ex14-1-2.c のように書き直すと, 末尾の行が２度出力される.
7. ex14-1-3.c のように scanf を利用するとどのようになるかを考えてみよう.
8. 「標準入力から２つの整数を読む」ために ex14-1-4.c のようなプログラムを書いたとしよう. このプログラムに “a␣b” というデータを入力したとき, n, m の値がどうなるかは規定されていない.
   このように scanf を利用してデータを読み込むと, そのデータが「正しい入力かどうか」の判定ができないため, scanf **またはその類似の関数を利用してデータ読み込みを行ってはならない.** scanf は printf で出力された「フォーマットの確定したデータ」を読むための関数であり, フォーマットの確定しないデータを読むためのものではない.

★ ex14-2.c **ファイルからのテキストデータの読み込み**.

1. ex14-1-1.c の改良版である. 「標準入力」からではなく, 指定のファイルからテキストデータを読み込む.
2. そのための違いは以下の通りである.
   - ファイルを指定し, ファイルを開くための関数 fopen を用いる. fopen の戻り値は, そのファイルにアクセスするための「ファイルポインタ」である. ファイルポインタは FILE * 型（これは stdio.h で定義されている）である.
     ここでの「ファイルポインタ」という用語は, 上に出てきた「ファイルポインタ」とは異るものである.
   - feof, fgets 関数の stdin のかわりに fopen で得られたファイルポインタを用いる.
   - ファイルの読み込みが終了したら fclose でファイルを閉じる必要がある.
3. fopen は２つの引数をとる. 第一引数は「ファイルパス」であり, 第二引数はファイルを開くモードの指定である. この場合「読み込み専用」であるので "r" を指定する.
   第一引数に指定したファイルが存在しない場合には NULL を返す.

★ ex14-3.c **ファイルへのテキストデータの書き出し:（新規作成モード）**.

1. ex14-1-2.c の改良して, ファイルからテキストデータを読み, 異るファイルに書き出しをしてみよう. この場合は, 読み込みファイルと, 書き出しファイルの両方を開くことが必要となる.
2. 書き込み用（新規作成）としてファイルを開くためには, fopen の第二引数に "a" を指定する.（fopen の仕様を調べてみよう. 他にどのようなモードがあるのだろうか？）
3. 出力は printf のかわりに fprintf を使えばよい.

★ ex14-4.c **ファイルからのバイナリデータの読み込みと書き出し**.

1. 今度は「必ずしもテキストファイルとは限らないファイル」の読み込みと書き出しを行ってみよう.
2. テキストデータは fgets で読み込みができ, fprintf で書き出しが可能である. しかし, 明らかに printf ではバイナリデータの出力ができないことがわかる.
   また, fgets では読み込んだデータの末尾に NULL 文字を付加するため, バイナリデータの読み込みがうまくいかない.
3. そのため, 読み込みには fread 関数, 書き出しには fwrite 関数を用いる必要がある.
   fread 関数は読み込んだバイト数を返すので, その量だけを fwrite 関数で書き込めばよい.

★ ex14-5.c **テキストデータの読み込みとデータの解析**.

1. ここでは, 標準入力からファイルを読み込み, そこに含まれる「行数」, 「文字数」を表示するプログラムを作ってみよう.
   これは UNIX の標準コマンド wc のオモチャである.
2. ただし, プログラムを簡単にするため, 入力されるファイルの１行の文字数は MAX_CHAR_PAR_LINE - 1 を越えないことを仮定する.

【課題】

★ exercise-14-1 ex14-5.c を次のように改良しなさい.

☆ exercise-14-1-1 入力されるテキストファイルに含まれる「単語数」も数えるようにしなさい. ただし, 「単語」とは「isspace が 0 でない値を返す文字を区切り文字とするトークン」のこととします.（すなわち, "␣\n\t\v\r" を区切り文字とします.）

【ヒント】 exercise-12-6 で作った _strtok 関数, または, 文字列操作関数 strtok を使えばよい.

☆ exercise-14-1-2 （**難**）exercise-14-1-1 を改良し, 入力されるファイルの１行の文字数が MAX_CHAR_PAR_LINE - 1 を越えてもよいようにしなさい. ただし, １単語の長さは MAX_CHAR_PAR_LINE - 1 を越えることは無いとします.

【ヒント】 読み込みごとに１行を完全に読み取ったかどうかを判定する必要があります.

★ exercise-14-2 標準入力からテキストデータを１行読み込み, そこに書かれた「空白で区切られた最大１０個の整数」の和を求めるプログラムを書きなさい.

ただし, テキストデータ中に「整数」ではない文字列が書かれている場合や, １０個を越える「整数」が書かれている場合には, 正しい結果を返す必要はありません.

【入力例】

- “0␣1␣−2␣+3␣␣4␣5␣−6␣␣␣7␣8␣␣␣+10” という入力を受け取った場合には, 整数 0, 1, −2, 3, 4, 5, −6, 7, 8, 10 が入力されたものとみなします.

- “␣␣0␣1␣−2␣+3␣␣4␣5␣−6␣␣␣7␣8␣␣␣+10” という入力を受け取った場合には, 整数 0, 1, −2, 3, 4, 5, −6, 7, 8, 10 が入力されたものとみなします. すなわち, 先頭に空白がある場合があります.

★ exercise-14-3 標準入力からテキストデータを読み込み, １行ごとに書かれた「空白で区切られた最大１０個の整数」の和を順に表示するプログラムを書きなさい.

ただし, テキストデータ中に「整数」ではない文字列が書かれている場合や, １行に１０個を越える「整数」が書かれている場合には, 正しい結果を返す必要はありません. （前の問題は「１行しか見ない」が, この問題は「各行ごと」に結果を返すという違いがあります.）

★ exercise-14-4 次の仕様をみたすプログラムを書きなさい.

【形式】

```
cat [file....]
```

【機能説明】

プログラム cat は引数に与えられたテキストファイルを順に読み込み, それらを標準出力に出力します.

【利用例】

- `cat file1 file2`
  file1, file2 の中身を連結した結果を標準出力に出力します.

★ exercise-14-5 次の仕様をみたすプログラムを書きなさい.

【形式】

```
cp file1 file2
```

【機能説明】

プログラム cp は file1 を file2 にコピーします. 引数が２個でない場合にはエラーと判断し, 戻り値 -1 を返します. file1 が存在しない場合にはエラーと判断し, 戻り値 -2 を返します.

【注意】

file2 が存在している場合には, それを上書きします.

★ exercise-14-6 次の仕様をみたすプログラムを書きなさい.

【形式】

```
od file
```

【機能説明】

プログラム od は file の内容を 16 進ダンプします. すなわち, file のバイト列を先頭から表示します.

【表示方法】

```
00000000: 20 21 22 23  24 25 26 27  28 29 0a 31  32 33 34 35
00000010: 36 37 38 39  3a 3b 3c 3d  3e 3f 40 41  42 43 44 45
```

のように１行に 16 バイト分, 4 バイトごとに空白を余分に入れます.

★ exercise-14-7 exercise-14-4 を改良して, 次の仕様をみたすプログラムを書きなさい.

【形式】

```
cat [file....]
```


【機能説明】

プログラム cat は引数に与えられたテキストファイルを順に読み込み, それらを標準出力に出力します. 引数が無い場合には標準入力からデータを読み込みます.

【利用例】

- `cat file`
  file の中身を標準出力に出力します.
- `cat file1 file2`
  file1, file2 の中身を連結した結果を標準出力に出力します.
- `cat`
  標準入力からのテキストデータを標準出力に出力します.

★ exercise-14-8 （**難**[2]）標準入力から１行読み込み, そこに書かれた「算術式」を「逆ポーランド記法」に変換した結果を表示するプログラムを書きなさい. ここでは, 入力されたものが「算術式」であるかどうかのエラー判定は必要ないものとします.

ただし, 「算術式」とは「識別子」 (identifier), 「四則演算」をあらわす記号 (operator), 「演算の優先順位を変更する括弧 (,)」, 「空白文字」からなり, 通常の意味で「計算式」をあらわす文字列のこととします. 演算子 / は「商」をあらわすものとし, 単項の - は使わないものとします.

また, プログラムを簡単にするため, 「識別子」は１文字の英数字で表されることとします. 全ての数値・演算子・括弧の間には一つ以上の空白が入ることとします.

【算術式の例】

- “1␣+␣2␣*␣3” は算術式です.
- “(␣1␣+␣2␣)␣*␣3” は算術式です.
- “a␣+␣2␣*␣3” は算術式です.
- “(␣−1␣+␣2␣)␣*␣3” は算術式ではありません. なぜなら「単項の -」 が使われています.
- “(␣1␣+␣+␣2␣)␣*␣−3” は算術式ではありません. なぜなら 1++2 の + が２項演算子になっていません. また, *- という記述があります.

通常の算術式では a + b のように, 算術演算子が「中置」になっていますが, 逆ポーランド記法では, これを a b + と書き, 算術演算子を「後置」します.

【逆ポーランド記法の例】

- 算術式 “1 + 2 * 3” を逆ポーランド記法に書き直したものは “1 2 3 * +” となります.
- 算術式 “2 * 3 + 1” を逆ポーランド記法に書き直したものは “2 3 * 1 +” となります.
- 算術式 “(1 + 2) * 3” を逆ポーランド記法に書き直したものは “1 2 + 3 *” となります.
- 算術式 “3 * (1 + 2)” を逆ポーランド記法に書き直したものは “3 1 2 + *” となります.
- 算術式 “(1 - 2) / 3” を逆ポーランド記法に書き直したものは “1 2 - 3 /” となります.

このように逆ポーランド記法では「括弧」が必要なくなります.

【アルゴリズムのヒント】

算術式を逆ポーランド記法に変換するアルゴリズムは（本質的に同じですが）以下の２つの方法が考えられます.

1. トークンに優先順位を与え, スタックを操作する方法
   (a) トークンに対して以下の優先順位を与えます.

トークン	優先順位
(	4
数値（識別子）	3
乗法演算子	2
加法演算子	1
)	0

   (b) 読み出したトークンの優先順位がスタックトップの優先順位より小さい間, スタックトップを出力し, 読み出したトークンをスタックに積む.
   (c) トークンの読み出しが終了したとき, スタックを順に出力する.
2. 以下の算術式 (Expression) の定義をそのまま再帰で表現する方法.

```
<Expression> ::= <Term> | <Expression> <Additive Operator> <Term>
<Term>       ::= <Factor> | <Term> <Multiplicative Operator> <Factor>
<Factor>     ::= <Identifier> | ( <Expresssion> )
<Identifier> ::= <Alpha Numeric Character>
```

★ exercise-14-9 標準入力から１行読み込み, そこに書かれた「算術式」の値を表示するプログラムを書きなさい. ただし, この問題では Identifier は１桁の１０進数とします.

★ exercise-14-10 （**やや難?**）電子メールで添付ファイルを添付するときに用いられる「**Base64 Encoding**」とは次のようなものです.

【符号化】 （バイナリファイルをテキストファイルに符号化する）
バイナリファイルを先頭から順に 6 ビットずつとりだし, その 6 ビットのデータを 6 桁の２進数として,
`ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/`
という文字にマッピングする. すなわち, その 6 ビットのデータが `0x00` ならば, `A` とし, `0x3F` なら `/` とする. このマッピングを行なったデータを, 改行文字を除いて, １行あたり 76 文字のテキストファイルとする. ただし, 最終行は 76 文字に満たなくてもよい.
この時, ファイル末尾では 24 ビットに満たないまとまりが出ることがある. この時, 入力ファイルで最後に 8 ビットが余った場合には, 入力データに 8 ビットの 0 をつけ加え, 出力文字から 1 文字を削って, その代りに 1 つの = を付け加える. 16 ビットが余った場合には, 入力データに 16 ビットの 0 をつけ加え, 出力文字から 2 文字を削って, その代りに 2 つの = を付け加える.

【復号化】 （符号化されたテキストファイルをバイナリファイルに戻す）
これは「符号化」の逆操作である.

この Base64 Encoding の符号化を行うプログラム, その復号化を行うプログラムを書きなさい.

★ exercise-14-10 （**難**[3]）exercise-14-8 の算術式の定義に「巾乗演算子」“^” を付け加えて同様の問題を考えなさい.

【ヒント】 巾乗演算子は右結合を持つ, 最も優先順位が高い演算子です. その事実を「算術式の定義」にうまく組み込み, exercise-14-8 をちょっとだけ改良すればおしまいです.

【注意】 上記の問題では, 特に指定していない限り, １行に含まれる文字数は MAX_CHAR_PAR_LINE - 1 文字以下と仮定してかまいません.

【その他】 電子メールで「今日の講義の感想や意見」を送ってください.

★ ex14-1-1.c の内容

```
/* $Id: ex14-1-1.c,v 1.2 2004-07-12 14:39:34+09 naito Exp $ */
#include <stdio.h>
#define MAX_CHAR_PAR_LINE       (1024)
int main(int argc, char **argv)
{
  char  buf[MAX_CHAR_PAR_LINE] ;
  while(!feof(stdin)) {
    if (fgets(buf, MAX_CHAR_PAR_LINE, stdin) != NULL) printf("%s", buf) ;
  }
  return 0 ;
}
```

★ ex14-1-2.c の内容

```
/* $Id: ex14-1-2.c,v 1.2 2004-07-12 14:39:43+09 naito Exp $ */
#include <stdio.h>
#define MAX_CHAR_PAR_LINE       (1024)
int main(int argc, char **argv)
{
  char  buf[MAX_CHAR_PAR_LINE] ;
  while(!feof(stdin)) {
    fgets(buf, MAX_CHAR_PAR_LINE, stdin) ;
    printf("%s", buf) ;
  }
  return 0 ;
}
```

★ ex14-1-3.c の内容

```
/* $Id: ex14-1-3.c,v 1.2 2004-07-12 14:39:57+09 naito Exp $ */
#include <stdio.h>
#define MAX_CHAR_PAR_LINE       (1024)
int main(int argc, char **argv)
{
  char  buf[MAX_CHAR_PAR_LINE] ;
  while(!feof(stdin)) {
    scanf("%s", buf) ;
    printf("%s\n", buf) ;
  }
  return 0 ;
}
```

★ ex14-2.c の内容

```
/* $Id: ex14-2.c,v 1.2 2004-07-12 14:40:16+09 naito Exp $ */
#include <stdio.h>
#define MAX_CHAR_PAR_LINE       (1024)
int main(int argc, char **argv)
{
  char  buf[MAX_CHAR_PAR_LINE] ;
  FILE  *in ;
  if ((in = fopen("ex14-2.c", "r")) == NULL) return -1 ;
  while(!feof(in)) {
    if (fgets(buf, MAX_CHAR_PAR_LINE, in) != NULL) printf("%s", buf) ;
  }
  fclose(in) ;
  return 0 ;
}
```

★ ex14-3.c の内容

```
/* $Id: ex14-3.c,v 1.2 2004-07-12 14:40:28+09 naito Exp $ */
#include <stdio.h>
#define MAX_CHAR_PAR_LINE       (1024)
int main(int argc, char **argv)
{
  char  buf[MAX_CHAR_PAR_LINE] ;
  FILE  *in, *out ;
  if ((in = fopen("ex14-2.c", "r")) == NULL) return -1 ;
  if ((out = fopen("outfile", "a")) == NULL) return -1 ;
  while(!feof(in)) {
    if (fgets(buf, MAX_CHAR_PAR_LINE, in) != NULL) fprintf(out, "%s", buf) ;
  }
  fclose(out) ; fclose(in) ;
  return 0 ;
}
```


★ ex14-4.c の内容

```
/* $Id: ex14-4.c,v 1.2 2004-07-12 14:40:42+09 naito Exp $ */
#include <stdio.h>
#define MAX_CHAR        (1024)
int main(int argc, char **argv)
{
  char          buf[MAX_CHAR] ;
  size_t        size ;
  FILE          *in, *out ;

  if ((in = fopen("./a.out", "r")) == NULL) return -1 ;
  if ((out = fopen("./b.out", "a")) == NULL) return -1 ;
  while(!feof(in)) {
    if ((size = fread((char *)buf, sizeof(char), MAX_CHAR, in)) != 0)
      fwrite(buf, sizeof(char), size, out) ;
  }
  fclose(out) ; fclose(in) ;
  return 0 ;
}
```

★ ex14-5.c の内容

```
/* $Id: ex14-5.c,v 1.2 2004-07-12 14:40:55+09 naito Exp $ */
#include <stdio.h>
#include <strings.h>
#define MAX_CHAR_PAR_LINE       (1024)
int main(int argc, char **argv)
{
  char  buf[MAX_CHAR_PAR_LINE] ;
  int   lines = 0, letters = 0 ;
  while(!feof(stdin)) {
    if (fgets(buf, MAX_CHAR_PAR_LINE, stdin) != NULL) {
      lines   += 1 ;
      letters += strlen(buf) ;
    }
  }
  printf("%d %d\n", lines, letters) ;
  return 0 ;
}
```