第1回　プログラミングゼミ

programming semi

川崎 高志　佐野 正樹　奥田 環

ver.1.0　　2001.06.19

1　はじめに

私たちが研究活動に用いているプログラムには，いくつかのパラメータ設定が伴います．例えば，GA の交叉率や，SA のクーリング周期などがそうです．これらの値をソースコードで直接設定している場合，変更が必要になる度にリコンパイルしなければなりません．通常は，パラメータの値をいろいろ変えて数値実験を行うことが多いので，このような実装ではとても面倒です．

そこで，パラメータの値を記した設定ファイルを読み込み，それをプログラムに反映させるようにすると便利です．今回のプログラミングゼミでは，そのようなプログラムの実例を示し，そこで用いられているアルゴリズムについて説明します．また，そのプログラムを，オブジェクト指向という観点からも解析します．

2　仕様

今回用いたプログラムを資料として A に載せています．

2.1　設定ファイルの書式

データを読み込ませるための設定ファイルの書式について説明します．このファイルには，`valuename = value` のように，変数を記述します．変数名や変数（値）は，英数字であれば，”`value name`”のように，間にスペースが入っても問題ありません．また，”=”（イコール）の前後や`valuename` の前にスペースを入れることもできます．

コメントアウトするには，”`#`”（シャープ）または”`;`”（セミコロン）を用います．これら以降の文字は読み飛ばされます．

設定ファイルのサンプルを Fig. 1 に示します．

```
#-----GA parameter
    popsize = 200
    mutation rate = 0.01
    crossover rate = 1.0
    generation = 1000
;end
```

Fig. 1　sample.txt

2.2　プログラムの仕様

このプログラムでは，以下のようなことを実行するための関数が用意されています．

- 指定したファイルからデータを読み込み
- データの検索

  - 変数名を指定して値を返す
  - 整数 n を指定して，n 番目の変数名を返す
  - 整数 n を指定して，n 番目の変数（値）を返す
- データの書き換え
- 終了処理

変数（値）として，文字列を書き込むこともできます．しかし，変数（値）を int 型や double 型に変換する場合，その変数の型については，各自で管理してください．

3　実行例

ここでは，このプログラムの実際に使用してみます．設定ファイルとして Fig. 1 を用い，Fig. 2 に示すサンプルプログラムを実行します．実行結果を Fig. 3 に示します．

```
// sample program
#include <stdio.h>
#include <stdlib.h>
#include "loadconf.h"

main()
{
    int popsize = 0;
    double mutation_rate = 0.0;

    CONFFILE_T cf = conf_open("sample.txt");  // 設定ファイルの読み込み

    if(cf)
    {
        popsize = atoi(conf_read(cf, "popsize"));
        mutation_rate = atof(conf_read(cf, "mutation rate"));

        printf("population size = %d\n", popsize);
        printf("mutation rate = %.2f\n", mutation_rate);
    }

    conf_close(cf);   // 終了処理

    return 0;
}
```

Fig. 2　サンプルプログラム

```
population size = 200
mutation rate = 0.01
```

Fig. 3　実行結果

4　データ構造とアルゴリズム

4.1　データ構造

このプログラムでは，Fig. 4 に示すように，複数の変数名と変数（値）のデータ組が格納されてます，このデータ組の数は変数”nVar”に格納されています．このプログラムでのデータ構造の宣言部分を Fig. 5 に示します.

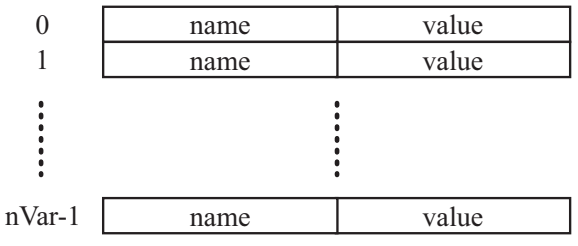

Fig. 4　データ構造

```
struct CONFFILE_CTX
{
    int nVars;       // number of variables
    struct VARINFO
    {
        char *name;  // variable name
        char *value; // variable value
    } *list;
};
```

Fig. 5　CONFFILE_CTX の定義

4.2　アルゴリズム

このプログラムのアルゴリズムを検索，格納，終了処理の 3 つに分けて説明します.

4.2.1　検索アルゴリズム

格納されているデータから，変数名，変数（値）を返します．具体的には次のような関数があります.

- `const char *conf_read(const CONFFILE_T conf, const char *varname)`
  変数名を指定すると，変数（値）を返します．この関数の検索では，データ数 n の場合，n 個を順に探していく，線形探索となります.
  （Fig. 6）

```
const char *conf_read(const CONFFILE_T conf, const char *varname)
{
    int i;
    for(i = 0; i < conf->nVars; i++)
        if(strcmp(conf->list[i].name, varname) == 0)
            return conf->list[i].value;
    return "";
}
```

Fig. 6　conf_read 関数

```
const char *conf_name(const CONFFILE_T conf, int n)
{
    return conf->list[n].name;
}
```

Fig. 7　conf_name 関数

- const char *conf_name(const CONFFILE_T conf, int n)
  n 番目の変数名を返します．データ量が増加しても，索引による検索となりますので，検索時間はどんな場合でも $O(1)$ です．
  （Fig. 7）

- const char *conf_value(const CONFFILE_T conf, int n)
  n 番目に格納されている変数（値）を返します．この関数も索引による検索ですから，検索時間はどんな場合でも $O(1)$ です．
  （Fig. 8）

4.2.2　格納アルゴリズム

ここでは，指定した読み込みファイルから，データを格納するアルゴリズムについて説明します．データ格納アルゴリズムを図示した Fig. 9 を示します．

データ格納は，conf_open 関数で行っています．データ格納のアルゴリズムを以下に示します．

1. 設定ファイルから 1 行ずつデータを読み込みます．
   freadline() 関数（Fig. 10，11）

```
const char *conf_value(const CONFFILE_T conf, int n)
{
    return conf->list[n].value;
}
```

Fig. 8　conf_value 関数

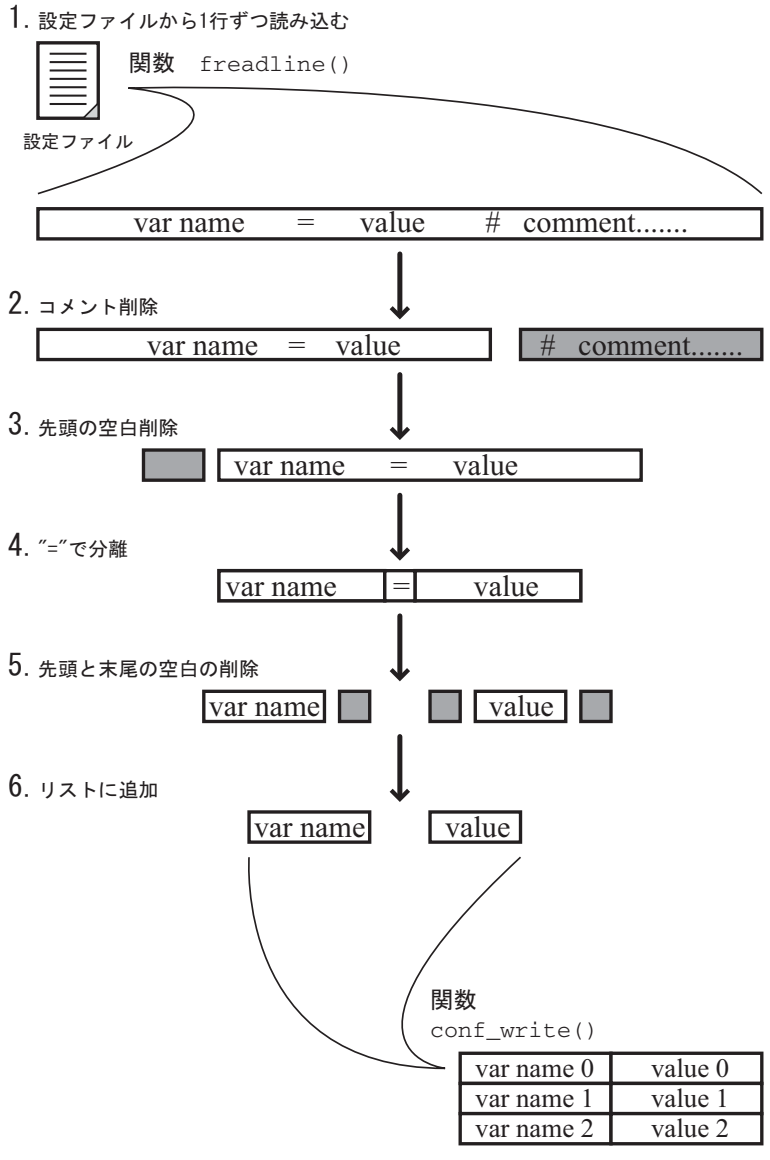

Fig. 9 格納アルゴリズム

この関数では，先頭の文字から改行文字を探しながら，文字を読み込んでいきます．改行文字または EOF を見つけた時点で，読み込みを終了します．

2. コメント部分を削除します．
読み込んだ 1 行の先頭文字から順に，"#"（シャープ）または";"（セミコロン）を探します．これらの文字を見つければ，その文字から後ろの文字列を削除します．

3. 先頭の空白を削除します．
`seek_word()` 関数（Fig. 12）
この関数では，先頭の文字が空白であれば，先頭からの空白部分を読み飛ばします．ここで空行判定もしています．

4. "="（イコール）で変数名と変数（値）を分離します．
空白部分を取り除いた，データ部分の先頭から"="を探し，そこを基準に分離します．もしなければ，`syntax error` になります．

5. 先頭と末尾の空白を削除します.

6. リストに追加します.
   `conf_write()` 関数
   この関数では，取り出した変数名と変数（値）をリストに追加します.

関数"`conf_open`"のプログラムを Fig. 13，Fig. 14 に示します.

4.2.3　終了処理

終了処理では，データの格納のために確保したメモリを解放します．プログラムを Fig. 15 に示します.

5　オブジェクト指向

オブジェクト指向として，継承・多態性・情報隠蔽等が上げられます．本節では，情報隠蔽に着目して，もう一度このプログラムを見ていきます.

5.1　情報隠蔽

このプログラムでは，以下のように情報隠蔽をしています.

- ヘッダファイルに実装部分を記述しない.
- static 関数を用いている.

このプログラムの`loadconf.h` は外部公開用として書かれています．つまり，このプログラムを使う上で必要な情報は書かれていますが，それ以上のことは書かれていません．ヘッダファイルに実装を書きたい場合は，ヘッダファイルを内部用と外部用に分けます.

Fig. 10-12 に書かれている関数，`freadline()`，`seek_word()` は，他の関数とは違ってヘッダファイルで宣言されていません．これらの関数は，関数ヘッダの先頭で`static` と記述されている，`static` 関数になっています.

`static` 関数は，他のファイルから呼び出すことができません．つまり，`static` 関数のスコープは，ファイルスコープになります.

`static` 関数にすることで，これらの関数が他のファイルから呼び出されることはありません．また，ユーザは，ヘッダファイルしか見ることができないので，これらの関数を知ることはなく，必要な部分だけを使用します.

```
static char *freadline(FILE *fp)
{
    const size_t ADDITION = 2000;
    char *buf = NULL;
    size_t size = 0;
    size_t cur = 0;
    int c;

    for(;;)
    {
        if(size <= cur)
        {
            char *temp = (char *)realloc(buf, sizeof(char) * (ADDITION + size));
            if(temp == NULL)
            {
                if(buf) free(buf);
                return NULL;
            }
            buf = temp;
            size += ADDITION;
        }
        c = fgetc(fp);
        if(c == EOF)
        {
            if(cur == 0)
            {
                free(buf);
                return NULL;
            }
            buf[cur] = 0;
            return buf;
        }
```

Fig. 10 `freadline` 関数 (1)

```
        else if(c == '\r')
        {
            c = fgetc(fp);
            if(c != '\n') // for Macintosh format text
                ungetc(c, fp);

            buf[cur] = 0;
            return buf;
        }
        else if(c == '\n')
        {
            buf[cur] = 0;
            return buf;
        }
        else
            buf[cur++] = c;
    }
}
```

Fig. 11　`freadline` 関数 (2)

```
static char *seek_word(char *p)
{
    p += strspn(p, SPACE_CHARS);
    if(*p == 0)
        return NULL;
    return p;
}
```

Fig. 12　`seek_word` 関数

```
CONFFILE_T conf_open(const char *fn)
{
    FILE *fp;
    char *str;
    int line;
    CONFFILE_T conf;

    // allocate instance
    conf = (CONFFILE_T)malloc(sizeof(struct CONFFILE_CTX));
    if(conf == NULL)
        return NULL;
    memset(conf, 0, sizeof(struct CONFFILE_CTX));

    // open file
    fp = fopen(fn, "rb"); // binary-mode
    if(fp == NULL)
    {
        free(conf);
        return NULL;
    }

    for(line = 1; str = freadline(fp); line++)
    {
        char *p, *q;
        char *varname;
        char *content;

        // remove trailing comments(#)
        p = strpbrk(str, "#;");
        if(p) *p = 0;
       // seek variable name
        varname = seek_word(str);
        if(varname)
        {
```

Fig. 13 conf_open 関数 (1)

```
            // seek equal(=)
            p = strchr(str, '=');
            if(p)
            {
                // split into varname-part and content-part
                *p = 0;
                content = p + 1;

                // find the end of varname
                for(p = varname; q = next_word(p); p = q);
                p = strpbrk(p, SPACE_CHARS);
                if(p) *p = 0;

                // seek content
                content = seek_word(content);
                if(content)
                {
                    // seek the last word
                    for(p = content; q = next_word(p); p = q);
                        p = strpbrk(p, SPACE_CHARS);
                    if(p) *p = 0;
                }
                else
                    content = ""; // empty

                conf_write(conf, varname, content);
            }
            else
            {
                fprintf(stderr, "LINE %d, no equal(=) found.\n", line);
                conf_close(conf);
                free(str);
                fclose(fp);
                return NULL;
            }
        }
        free(str);
    }
    fclose(fp);
    return conf;
}
```

Fig. 14　conf_open 関数 (2)

```
void conf_close(CONFFILE_T conf)
{
    if(conf)
    {
        int i;
        for(i = 0; i < conf->nVars; i++)
        {
            if(conf->list[i].name) free(conf->list[i].name);
            if(conf->list[i].value) free(conf->list[i].value);
        }
        if(conf->list) free(conf->list);
        free(conf);
    }
}
```

Fig. 15 conf_value 関数

A　プログラム

A.1　loadconf.h

```
//-----------------------------------------------------------------------------
// FILENAME    : loadconf.h
// DESCRIPTION : Loading Configuration Data
//-----------------------------------------------------------------------------
#if !defined(_LOADCONF_H_)
#define _LOADCONF_H_

struct CONFFILE_CTX;
typedef struct CONFFILE_CTX * CONFFILE_T;

CONFFILE_T conf_open(const char *fn);
void conf_close(CONFFILE_T conf);
int conf_write(const CONFFILE_T conf, const char *varname, const char *value);
const char *conf_read(const CONFFILE_T conf, const char *varname);
const char *conf_name(const CONFFILE_T conf, int n);
const char *conf_value(const CONFFILE_T conf, int n);
int conf_nvars(const CONFFILE_T conf);

#endif /* _LOADCONF_H_ */
```

A.2　loadconf.h

```
//-----------------------------------------------------------------------------
// FILENAME    : loadconf.c
// DESCRIPTION : Loading Configuration Data
//-----------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "loadconf.h"

struct CONFFILE_CTX
{
    int nVars;       // number of variables
    struct VARINFO
    {
        char *name;  // variable name
        char *value; // variable value
    } *list;
};

const char *SPACE_CHARS = " \t";
const char *SPACE_AND_EQUAL_CHARS = " \t=";
```

```
/* seek the head of the first word */
static char *seek_word(char *p)
{
    p += strspn(p, SPACE_CHARS);
    if(*p == 0)
        return NULL;
    return p;
}

/* the next word if available, NULL for nothing to be found */
static char *next_word(char *p)
{
    p = strpbrk(p, SPACE_CHARS);
    if(p == NULL) return NULL;
    p = p + strspn(p, SPACE_CHARS);
    if(*p == 0) return NULL;
    return p;
}

/* read a line from file */
static char *freadline(FILE *fp)
{
    const size_t ADDITION = 2000;
    char *buf = NULL;
    size_t size = 0;
    size_t cur = 0;
    int c;

    for(;;)
    {
        if(size <= cur)
        {
            char *temp = (char *)realloc(buf, sizeof(char) * (ADDITION + size));
            if(temp == NULL)
            {
                if(buf) free(buf);
                return NULL;
            }
            buf = temp;
            size += ADDITION;
        }

        c = fgetc(fp);
        if(c == EOF)
        {
```

```
            if(cur == 0)
            {
                free(buf);
                return NULL;
            }
            buf[cur] = 0;
            return buf;
        }
        else if(c == '\r')
        {
            c = fgetc(fp);
            if(c != '\n') // for Macintosh format text
                ungetc(c, fp);

            buf[cur] = 0;
            return buf;
        }
        else if(c == '\n')
        {
            buf[cur] = 0;
            return buf;
        }
        else
            buf[cur++] = c;
    }
}

void conf_close(CONFFILE_T conf)
{
    if(conf)
    {
        int i;
        for(i = 0; i < conf->nVars; i++)
        {
            if(conf->list[i].name) free(conf->list[i].name);
            if(conf->list[i].value) free(conf->list[i].value);
        }
        if(conf->list) free(conf->list);
        free(conf);
    }
}

CONFFILE_T conf_open(const char *fn)
{
    FILE *fp;
    char *str;
```

```
int line;
CONFFILE_T conf;

// allocate instance
conf = (CONFFILE_T)malloc(sizeof(struct CONFFILE_CTX));
if(conf == NULL)
    return NULL;
memset(conf, 0, sizeof(struct CONFFILE_CTX));

// open file
fp = fopen(fn, "rb"); // binary-mode
if(fp == NULL)
{
    free(conf);
    return NULL;
}

for(line = 1; str = freadline(fp); line++)
{
    char *p, *q;
    char *varname;
    char *content;

    // remove trailing comments(#)
    p = strpbrk(str, "#;");
    if(p) *p = 0;

    // seek variable name
    varname = seek_word(str);
    if(varname)
    {
        // seek equal(=)
        p = strchr(str, '=');
        if(p)
        {
            // split into varname-part and content-part
            *p = 0;
            content = p + 1;

            // find the end of varname
            for(p = varname; q = next_word(p); p = q);
            p = strpbrk(p, SPACE_CHARS);
            if(p) *p = 0;

            // seek content
            content = seek_word(content);
```

```
                if(content)
                {
                    // seek the last word
                    for(p = content; q = next_word(p); p = q);
                    p = strpbrk(p, SPACE_CHARS);
                    if(p) *p = 0;
                }
                else
                    content = ""; // empty

                conf_write(conf, varname, content);
            }
            else
            {
                fprintf(stderr, "LINE %d, no equal(=) found.\n", line);
                conf_close(conf);
                free(str);
                fclose(fp);
                return NULL;
            }
        }
        free(str);
    }
    fclose(fp);
    return conf;
}

int conf_write(const CONFFILE_T conf, const char *varname, const char *value)
{
    struct VARINFO *temp;
    int i;

    for(i = 0; i < conf->nVars; i++)
        if(strcmp(conf->list[i].name, varname) == 0)
        {
            free(conf->list[i].value);
            conf->list[i].value = strdup(value);
            return 0;
        }

    // memory reallocation
    temp = (struct VARINFO *)realloc(
        conf->list, sizeof(struct VARINFO) * (conf->nVars + 1));
    if(temp == NULL)
        return -1;
```

```
    conf->list = temp;

    // copy strings
    conf->list[conf->nVars].name = strdup(varname);
    conf->list[conf->nVars].value = strdup(value);
    conf->nVars++;

    return 0;
}

const char *conf_read(const CONFFILE_T conf, const char *varname)
{
    int i;
    for(i = 0; i < conf->nVars; i++)
        if(strcmp(conf->list[i].name, varname) == 0)
            return conf->list[i].value;
    return "";
}

int conf_nvars(const CONFFILE_T conf)
{
    return conf->nVars;
}

const char *conf_name(const CONFFILE_T conf, int n)
{
    return conf->list[n].name;
}

const char *conf_value(const CONFFILE_T conf, int n)
{
    return conf->list[n].value;
}
```

B ヘッダの 2 度読み防止

ヘッダ (loadconf.h) を見ると，その先頭と末尾に次のような文字列があることがわかります.

```
#if !defined(_LOADCONF_H_)          /*  (1)  */
#define _LOADCONF_H_                /*  (2)  */
            .
            .
            .
#endif /* _LOADCONF_H_ */          /*  (3)  */
```

これは，このヘッダを複数回読み込まないための措置です．簡単に説明すると，次のような処理が行われます.

1. 最初にヘッダが読み込まれると，マクロ_LOADCONF_H_が define されていないので，(1) の if 文が真と

なる．

2. (2) の行でマクロ_LOADCONF_H_が define される．

3. 2 回目にヘッダが読み込まれると，マクロ_LOADCONF_H_はすでに define されているので，(1) の if 文は偽となり，(3) にジャンプします．よって，(2) と (3) に挟まれた部分は読み飛ばされます．

マクロの名前は何でも良いのですが，他のマクロと同じ名前を使用しないように注意しなければなりません．慣習的に，ヘッダファイル名をマクロ名にする (loadconf.h なら，_LOADCONF_H_) ことが多いようです．

では，ヘッダを複数回読むと何が起こるのでしょう？それについて，Fig. 16 および Fig. 17 を用いて説明します．Fig. 16 では，意図的にヘッダ (header.h) を 2 回読み込んでいます．これをコンパイルすると，Fig. 18 のようなエラーが出ます．原因は，ヘッダを 2 回読み込んだことにより，STRUCT という構造体を 2 回定義したのと同じことになることです．このように，ヘッダの 2 度読み防止を行っていないと，文法エラーの原因になります．正しく書き直したヘッダを，Fig. 19 に示します．

```
#include "header.h"
#include "header.h"  /* 2 回読み込む */

int main()
{
    return 0;
}
```

Fig. 16　header.h を 2 回読み込む (main.c)

```
struct STRUCT
{
    int a, b;
};
```

Fig. 17　2 度読み防止の無いヘッダ (header.h)

```
In file included from main.cpp:2:
header.h:2: redefinition of ‘struct STRUCT’
header.h:4: previous definition here
```

Fig. 18　2 度読みによるコンパイルエラー

C　プラットフォームと改行コード

Windows・Macintosh・UNIX では，テキストファイルに使用している改行コードが全て異なります (Table 1)．今回のようにテキストファイルを読み込むプログラムでは，この点に関しても留意する必要があります．

```
#if !defined(_HEADER_H_)
#define _HEADER_H_

struct STRUCT
{
    int a, b;
};

#endif
```

Fig. 19　2 度読み防止を行っているヘッダ (header.h)

プラットフォーム	改行コード
Windows	[CR][LF]
UNIX	[LF]
Macintosh	[CR]

Table 1　プラットフォームと改行コード

今回紹介したプログラムでは freadline 関数で改行コードの識別を行っています．より具体的には，Fig. 20 に示す部分です．C 言語では，キャリッジリターン (CR) が'\r' に，ラインフィード (LF) が'\n' にそれぞれ対応しています．アルゴリズムは以下のとおりです．

1. 現在読み込んだ文字が'\r' の場合，次の文字を読む．次の文字が'\n' でなければ Macintosh 形式なので，ストリームに 1 文字戻す．'\n' ならば Windows 形式である．この判定の後，読み込み用バッファの現在位置をヌルコード ('\0') で上書きして，読み込みを終了する．
2. 現在読み込んだ文字が'\n' の場合 (UNIX 形式)，読み込み用バッファの現在位置をヌルコードで上書きして，読み込みを終了する．

D　typedef を用いる理由

紹介したプログラムでは，ユーザの使用する関数 (ヘッダ loadconf.h で宣言されている関数) は，すべて CONFFILE_T 型の変数を引数にとります．CONFFILE_T は，実際には **typedef 指定子** によって，CONFFILE_CTX 構造体へのポインタの別名として宣言されています．このようにまわりくどいことを行っている理由としては，次のことが挙げられます．

仕様を変更せずに実装を変更できる　CONFFILE_T という新しい型を導入したことにより，外部とのインタフェースを変更せずに，プログラムを修正・改良することができます．例えば，CONFFILE_T を整数型の ID に変更したとしても，クライアントプログラムを変更する必要はありません．データの格納形式をCONFFILE_CTX 構造体から変更したとしても同様です．

実装の詳細をユーザに意識させたくない　CONFFILE_T 型がポインタ型であることユーザに意識させると，ポインタに対する演算や処理 (インクリメントや free 関数によるメモリ開放など) をCONFFILE_T 型の変数に適用できると勘違いしてしまいます (現時点ではできるのですが)．このプログラムの仕様では，そのようなことを

```
else if(c == '\r')
{
    c = fgetc(fp);
    if(c != '\n') // for Macintosh format text
        ungetc(c, fp);

    buf[cur] = 0;
    return buf;
}
else if(c == '\n')
{
    buf[cur] = 0;
    return buf;
}
```

Fig. 20　freadline 関数の一部

保証していません．ユーザが，保証外のことを行って不利益を被ったとしても，それはユーザの責任です．しかし，できるだけ不正なことを行えないような仕様にしておくことは，開発者のマナーと言えます．

E　バッファオーバーランの防止

freadline 関数には，Fig. 21 のようなコードがあります．このコードでは，1 行読み込み用バッファの長さが不足しているかどうかを判定し，必要に応じてバッファのメモリを再確保しています．

```
if(size <= cur)
{
    char *temp = (char *)realloc(buf, sizeof(char) * (ADDITION + size));
    if(temp == NULL)
    {
        if(buf) free(buf);
        return NULL;
    }
    buf = temp;
    size += ADDITION;
}
```

Fig. 21　freadline 関数の一部 (バッファオーバーランの防止)

バッファを固定長にした場合，それがどんなに長くても，その長さを超えるデータが入力される可能性があります．バッファよりも長いデータを無理に入力した場合，Segmentation fault が発生したり，メモリの内容が不正に書きかえられたりすることになります．このような現象を **バッファオーバーラン** といいます．Fig. 21 に示したコードでは，このバッファオーバーランを防止しています．