プログラムのオプションを処理する方法について

平　野　照比古*

A Consistent Method to Deal with Program's Options

Teruhiko HIRANO

Abstract

Sometimes we want to execute some program under slightly different conditions from the original ones. There are several methods to do so: the program demands all the parameters when it starts. This method is troublesome when the program need a lot of parameters. Another method is to give it the only parameters which one wants to change as options. In this paper, we propose a consistent method to deal with options and show an example.

1. オプションの与え方の例

よく使われているプログラムで，オプションはどのように与えられているかをみよう。ここでは比較のためにMSDOSとUnixの場合を挙げる。取り扱うプログラムはファイルをリストアップするコマンドDIR（MSDOSの場合）とls（Unix）の場合である。

1.1 MSDOSのDIRの場合[6]

この場合単に

DIR

と入力すれば現在いるディレクトリーのファイル名とその大きさ，作成された日時等が表示される。このとき対象となっているファイルの数が多いと画面から溢れてしまい表示を途中で止める必要がある。オプションを利用しなければCTRL+Sを入力して止めるが，この方法ではタイミングがずれるとやり直しである。これを避けるためにはオプション/Pを用いればよい。つまり，

DIR /P

と入力すればよい。（Pは Pauseの略と思われる）DIRにはこの他にファイル名だけを横に5つずつ並べて表示する/Wというオプションがある。両方のオプションを利用したいときは

DIR /W /P

と入力する。また，あるディレクトリー（EXAMPLEとする）だけを表示したいときはDIR EXAMPLEと入力すればよい。これにオプションを付ける場合はオプションを適当な場所に書けばよい。つまり，

DIR /P EXAMPLE
DIR EXAMPLE /P

でもよい。もちろん，

DIR/P EXAMPLE/W

も許される。オプションによりプログラムの出力結果の形式を変化させることができる。

MSDOSにおけるオプションの与え方の特徴は次の通りである。

- 文字'/'はこの部分がオプションであることを示す。
- オプションを与える位置は制限がない。
- 複数のオプションは個別に与える。

1.2 Unixのlsの場合

lsはMSDOSの場合のDIR /Wに（ほとんど）等しい。Unixでは'.'（ピリオド）で始まるファイル名は

1993年9月24日受理

* 一般科

```
#include "SRC/optiondef.h"
extern char *ConfigFile;
static char * FindOptionAndValue( char *line );
static void SearchAndSet( char * Name, char *Name1 );
#define MaxChar    200
static OPTION *options;
static int OptionNumber ;
static Bool *override;

void SetOption( int argc, char *argv[], OPTION opts[], int OpNo,
                char *ConfigOptName, Bool *OverRide )
{
    int i; FILE *fp; char *OptName; char line[ MaxChar ];
    options = opts; OptionNumber = OpNo; override = OverRide;
    for( i=0; i< argc ; i++ ) {
        if( *(argv[i]) == '-' ) {
            strcpy( line, argv[i]+1);
            OptName = line;
            while( *OptName ) {
                if( *OptName++ == '=' ) {
                    *(OptName-1) = '\000'; break;
                }
            }
            if( strcmp( line, ConfigOptName ) == 0 ) {
                if( *OptName ) ConfigFile = OptName;
                argv[i] = "-remark"; break;
            }
        }
    }
    if( ConfigFile ) {
        if( (fp = fopen( ConfigFile, "rt" ))== NULL )
                errors("Configuration file %s not found.", ConfigFile);
        while( fgets( line, MaxChar, fp ) ) {
            if( line[0] == '-' )
                SearchAndSet( line+1, FindOptionAndValue( line+1 ));
        }
        fclose( fp );
    }
    for( i=0; i< argc ; i++ ) {
        if( *(argv[i]) == '-' )
            SearchAndSet( argv[i]+1, FindOptionAndValue( argv[i]+1 ));
    }
}
```

図1. オプションを設定するための関数

隠しファイルであるのでこのコマンドでは表示されない。また，ファイル名はアルファベット順に並べ変えられる。MSDOS の DIR と同じことをするためには

ls -l

と入力する。このとき，隠しファイルも同時にみるためには

ls -al

または

ls -a -l

などと入力する。(l は long, a は all の略)

Unix におけるオプションの特徴をまとめれば次の通りである。

- オプションは文字 '-' で始まる。
- オプションの名称は一文字であることが多い。
- 複数のオプションはつなげて書いてよい。
- 伝統的にはオプションはコマンド名に続いて書く必要がある。つまり

ls -al EXAMPLE

は正しいコマンドの与え方であるが，

ls EXAMPLE -al

は正しくない。これはパラメーマを取るオプションのときには空白を明けた次の文字列がそのパラメータになることに原因があると思われる。

1.3 GNU の場合

GNU とは Free Soft ware Foundation（FSF）がすすめるプロジェクトの名称で，その社長が Richard Stallman である。GNU ではコマンドが違っても同じ名前のオプションは同じ意味を表すようにするということである[5, p. 140]したがって，ここではオプションの名称は非常に長いものがある。実際，GNU プロジェクトの中で一番有名なプログラムである gcc（GNU C コンパイラー）では

-fomit-frame-pointer
-fstrength-reduce
-fwritable-strings

などのような長いオプション名が存在する。

2. オプションのデータ構造

これから，我々のオプションを処理する統一的なプログラムを提案するわけであるが，そのためにはいくつかの仕様を定めなければならない。今回使用する言語はオプションの処理が比較的に楽にできる C を用いる。C ではコマンドラインに書かれたコマンドに与えられたパラメータの処理が統一されているという点が有利である。他の言語，たとえば FORTRAN ではこのようなところが言語仕様としては与えられていないので，システムサブルーチンを利用することになり，機種間の互換性が低くなる。

2.1 オプションのとり得る値

上に挙げたオプションの例はすべてある動作をするように指示をしているので，オプションは論理型の値をとる変数とみることができる。ファイルをプリンターに出力するとき 1 ページに印刷する行数を指定するオプションを考えるとこれは正の整数の値をとる変数である。このようにオプションによっては取り得る値が異なる。ここでは次の値を考えることにする。

論理値　真または偽の値をとるオプションである。指定することにより既定植(デフォルト)と逆の値を取る。既定値の真，偽に応じてクラスを分けることが考えられる。

整数値　整数の値をとるオプションである。正の値しかとらないものと負の値もとることができるクラスに分けてもよい。また，範囲を指定することも可能である。

文　字　一文字の値をとるオプションである。整数型の場合の特別なこともみなせる。

文字列値　文字列を値としてとるオプションである。プリンターに出力するときにタイトルを与える場合等に利用できる。

2.2 コマンドラインにおけるオプションの与え方

Unix 流のオプションの与え方はここでは採用しない。つまり，オプションの名称は一文字とは限らない。上に述べたように，オプションは数字だけとも限らないし，文字列もくる可能性があるので，オプションの名称とその値の間にはなにかしらの区切り符号が必要である。c の言語仕様ではコマンドラインにおける空白は別のパラメーターとして扱われるので[1, 139 ページ]ここではオプション名称とその値の区切り符

表1. オプションを定義するための構造体の例

```
 1: typedef enum { FALSE, TRUE } Bool;
 2: typedef enum {
        IGNORE, BOOL, NBOOL, INT, UINT, REAL, CHAR, STRING } TYPE;
 3: typedef struct {
 4:     TYPE type;       /* data type of option */
 5:     char * name;     /* name of option */
 6:     union {
 7:         void *          NeverRefferred;
 8:         int             int_value;
 9:         unsigned int    uint_value ;
10:         char            char_value;
11:         char*           string;
12:         Bool            bool_value;
13:         double          real_value;
14:     } U;
15:     char * message;
16: } OPTION;

17: void SetOption( int , char *[] , OPTION [], int, char *, Bool * ) ;
18: void PrintOptionValue( OPTION[], int );
19: void Help( OPTION[], int, char * );
```

号として空白は採用しない。そこで次の形にする。

-〈オプション名〉［=〈オプションの値〉］

上の［…］の部分は省略可能であることを示している。この方法によりオプション名は＝と空白を含まない任意の文字列を利用することができる。

2.3 オプションの構造体

Cでは既存のデータを組み合わせて新たなデータ構造— 構造体を作ることができる［1, 第6章］ここでは，オプションのための構造体を考えよう。我々はオプションの統一的な処理を目的としているからオプションはある配列に格納することを考えている。したがって，各オプションを記述するためには名称，データタイプと値が必要である。また，簡単なオンラインマニュアルも出力できるようにするためにそのオプションの説明文（Help メッセージ）も必要であろう。このことを考慮すれば表1のような構造体を考えることができる。このプログラムの簡単な解説をしておく。

- 1行目は論理値の定義である。範囲を明確にするためにenum（列挙型）を用いている。［1, 48 ページ］
- 2行目はオプションが利用することのできるデータの型をやはりenumを用いて定義している。
  -IGNOREは値が利用されないことを示している。
  -BOOLとNBOOLはそれぞれ既定値が真と偽の論理データである。
  -INTとUINTはそれぞれ整数と正の整数のデータ型を意味する。
  -REALは実数のデータを意味する。
  -CHARは文字定数（一文字）を扱う。
  -STRAINGは文字列のデータ型を扱う。
- 3行目からは各オプションが保持するデータの構造体を定義している。いろいろなデータを扱うので，union（共用体）［1, 178～180 ページ］を用いてそのデータをしまう場所を確保している。7行目でvoid*があるがこれは共用体はその初めのデータ型でしか初期化できないというcの仕様のためである。［1, 180 ページ］（しかし，これでもdoubleのデータは初期化できない。）
- 15行目はHelpメッセージを出力するためのもの

表2.　オプションを定義する例

```
#include "optiondef.h"
OPTION options[] = {
    { /* InMode */ STRING, "inmode", {(void*)("c0")},
          "入力ファイルの形式 'c' 文字形式と初めの文字,\n"
          "\t\t'b'バイナリーモードとビット数(1,2,4,8,16)" },
    { /* OutMode */ STRING, "outmode", {(void*)("cA")},
          "出力ファイルの形式 'c' 文字形式と初めの文字,\n"
          "\t\t'b'バイナリーモードとビット数(1,2,4,8,16)" },
    { /* Width */ UINT, "width", {(void *)80},
          "横の大きさ、単位は文字数" },
    { /* Height */ UINT,  "height", {(void *)30},
          "縦の大きさ、単位は文字数" },
    { /* Repeated */ UINT, "rwidth", {(void *)10 },
          "繰り返しの大きさ単位は文字数"},
    { /* NoPatterns */ UINT, "pn", {(void *)26 } ,
          "使用するパターンの数" },
    { /* OverRide */ BOOL, "override", {(void *)FALSE},
          "ファイルのフォーマットよりオプションの指定を優先する."},
    { /* CommandLine */ STRING, "config", {(void *)"config" },
                /* option name    and its value must be same. */
            "コンフィグレーションファイルを読む"},
    { /* Remark */ STRING, "remark", {(void *)("")}, "注釈"},
    { /* FF */   STRING, "FF", {(void *)("\\x0f")},
          "改ページ制御文字列"},
    { /* CRLF */ STRING, "CRLF", {(void *)("\\x0a")},
          "改行制御文字列"},
    { /* DG */ INT, "debug", {(void *)0}, "Debug Mode"},
    { /* HP */ BOOL, "help", {(void *)FALSE}, "help"}
};
#define OptionNumber (sizeof(options)/sizeof(OPTION))

#include "MkOptvar.h"
```

である。

- 17行目から19行目まではこの仕様でオプションを扱うための関数の前宣言である。具体的な内容は後で述べる。

2.4　オプションの定義

各プログラムでオプションを定義したいときは表1をincludeしてその後で

OPTION option [ ]＝｛
　｛……｝,
　……
　｝；

なる初期化を利用してオプションを定義すればよい。しかし，この方法には重大な欠陥がある。オプションで定義した値が配列の変数としてしか参照できないことである。これは次のような問題を含む。

1. オプションの値をoption [n]. U. int_valueのようにしか参照できないのでプログラムの可読性が低下する。
2. オプションを追加すると他のオプションの値の

表3. オプションの値を参照するための例

```
#define  InMode options[ 0 ].U.string
#define  OutMode options[ 1 ].U.string
#define  Width options[ 2 ].U.uint_value
#define  Height options[ 3 ].U.uint_value
#define  Repeated options[ 4 ].U.uint_value
#define  NoPatterns options[ 5 ].U.uint_value
#define  OverRide options[ 6 ].U.bool_value
#define  CommandLine options[ 7 ].U.string
#define  Remark options[ 8 ].U.string
#define  FF options[ 9 ].U.string
#define  CRLF options[ 10 ].U.string
#define  DG options[ 11 ].U.int_value
#define  HP options[ 12 ].U.bool_value
#ifdef OptionNumber
#undef OptionNumber
#else
extern OPTION  options[];
#endif
#define OptionNumber  13
```

参照の方法が変わる可能性がある。一番最後に追加をすれば問題は避けられるが関連するオプションは近くに定義しておいた方がよい。

1を避けるためには#define文を利用して別の名称を定義すればよい。しかし，2の問題を避けることはできない。そのためにはなんらかの前処理をこのオプションを定義するファイルにして，オプションの変数の値を別称で定義するファイルを作成し，それをコンパイルするファイルに取り込めばよい。ここで提起する方法を例でもって示そう。表2はオプションを定義しているファイルの例である。各オプションを初期化する部分のところに注釈が入っているところに注意してほしい。たとえば，3行目では /* InMode */ という部分である。この部分と次のSTRINGをみて別のファイル（この場合では30行目にあるMKOptvar.hというファイル）に

#define InMode options [0]. U. string

という行を出力するようにすればよい。全体では表3のようになる。今回はこのような処理をするためにcで新たなプログラムを組むことはしなかった。このような処理をするのにはAWK [2, 3]が便利である。表4にそのプログラムを示す。

なお，オプションを定義するファイル（表2）が書き直されたら自動的に表3のファイルも更新される必要がある。今回はプログラム保守ユーテリティmakeを用いてこの部分を処理したのでは問題なかった。

3. オプションの処理

このように仕様が定まればオプションを処理するプログラムを書くことは易しい。ここでは今までに述べられていないいくつかの仕様の拡張について述べる。

3.1 仕様の拡張

今度のオプションを処理するに当たっては次の機能が追加さている

- オプションの既定値（デフォルト）を-configで与えたファイルから読み込むことができる。
- プログラムで処理されるファイルの先頭部にオプションを記述することができる。
- オプションの優先順位は次のようにした。
  1. 処理されるファイルに書いてあるオプションの値
  2. コマンドラインにおけるオプションの値
  3. 既定値

表4. オプションを処理するためのAWKプログラム

```
#
# option.h から変数名の定義のヘッダーファイルを作る
#
BEGIN{ c=0; }
$1 == "OPTION" {
    if( match( $2, /^(.)+\[\]$/ ) == 0 ) {
        print " not found OPTION name"; exit( 0 );
    }
    OptName = substr( $2, RSTART, RLENGTH-1);
}
$4 == "\*\/" {
    if($5~/^BOOL/) {
        print "#define ", $3, OptName, c++, "].U.bool_value";}
    else if ($5~/^INT/) {
        print "#define ", $3, OptName, c++, "].U.int_value";}
    else if ($5~/^UINT/) {
        print "#define ", $3, OptName, c++, "].U.uint_value";}
    else if ($5~/^REAL/) {
        print "#define ", $3, OptName, c++, "].U.real_value";}
    else if ($5~/^CHAR/) {
        print "#define ", $3, OptName, c++, "].U.char_value";}
    else if ($5~/^BOOL/) {
        print "#define ", $3, OptName, c++, "].U.bool_value";}
    else if ($5~/^STRING/) {
        print "#define ", $3, OptName, c++, "].U.string";}
}
END{
    print "#ifdef OptionNumber " ;
    print "#undef OptionNumber " ;
    print "#else";
    print "extern OPTION ", OptName"];" ;
    print "#endif";
    print "#define OptionNumber ", c;
}
```

したがって，オプションを処理するときには注意が必要である。まず，-configで指定したファイル(なければデフォルトのファイル名）に書いてあるオプション値を設定し，このオプションを無効にする。その後からコマンドラインのオプションの値を設定する。その後，処理されるファイルを読み，そこのオプションの値を設定する。場合によってはファイルの指定値を無視したいときもあるのでそのために-overrideというオプションを用意してある。これらのオプション名をすべてのプログラムで強制し，プログラミング上の問題もあるので，オプションを設定する関数には引数として渡すようにした。表1の17行目に前宣言してある関数SetOptionがそれである。各引数は次の意味を持つ。

int argc　コマンドラインから渡されたコマンドラ

イン上のパラメータの数

char *argv［ ］ コマンドラインから渡されたコマンドライン上のパラメータのリスト

OPTION options［ ］ 定義されているオプションのリスト

int Option No 定義されているオプションの数

char *ConfigOptName コンフィギュレーションファイルを指定するためのオプション名

Bool *override 処理されるファイルに書いてあるオプションを優先させないためのフラグを指すポインター

このプログラムリストを表1に示す。標準のヘッダーファイルは省略してある。この中にはいくつかの定義されていない関数が存在するがその機能は名称から明らかであろう。

4. 今後の課題

このようなオプションを統一的に扱う方法は他にもある。たとえば，川本[4]においてはプログラムの実行時にオプションを登録するようになっている。この方法では登録する関数にオプションの値をしまうアドレスを与えるのでオプションを追加するときに変更する場所が複数になるところが欠点であると思われる。ここにおける方法ではそれは避けることができる。現在，この方法で考えられる欠点は次の通りである。

- オプションについていくつかの特別な処理をしたのでオプションの処理する関数がライブラリーとして登録できない。
- 値の範囲や可能な値だけを指定する方法がない。
- 必ず登録しなければならないオプションが存在する
- # ifdef に対応してオプションを定義したりしなかったりすることが出来ない。

これらの問題解決にはオプションの構造体の定義や前処理の方法を変える必要がある。今後の問題として考えておきたい。

謝　辞

ここで述べたプログラムはすべて Sharp X68000 上で実行した。その際利用したソフトウェアは次の通りである。作成者並びに移植者に感謝する。

- GCC Ver 1.40 X6-25 X68000 真里子バージョン X6-25, 移植者　真理子氏
- みゅーへっだ Ver ξ-, (c) みゅーぷろじぇくと
- GNUlib
- X68k has — High-speed Assembler v2.50, 作成者 Y. Nakamura
- X68k hlk225 — SILK Hi-Speed Linker v2.25 作成者 SALT 氏
- Micro Emacs ver 3.10 J1.43 (rel. 3-)
- make, clib (XC ver. 2.0 に付属) Sharp
- jgawk 2.13.2+0.1 移植者　健ちゃん

参考文献

1) B.W. カーニハン，D.M. リッチー著(石田晴久訳) プログラミング言語 c 第2版 1989年，共立出版

2) エイホ，ワインバーガー，カーニハン著，(足立高徳訳)プログラミング言語 AWK，1989年，トッパン発行

3) D.B. Close, A.D. Robbins, P.H. Rubin, R. Stallman, The GAWK Manual, 1989, Free Software Foundation

4) 川本，*TEXpreviewer* と *TEXprinter ver 2.06t*, Nifty FSHARP3 フォーラムのデータライブラリー

5) きだあきら，*GNU* プロジェクトの過去・現在・未来，C マガジン，1933年3月号，136-141，ソフトバンク発行

6) 東芝発行 Book Computer DynaBook, DynaBook ガイド，平成2年