3N-4

OPTION：コマンドライン解析ルーチン自動生成プログラム

高濱　徹行

(福井大学工学部)

1．はじめに

C言語はコマンドライン・パラメータの引渡し方法が言語仕様として規定されているため、コマンド記述用として広く使用されている。

コマンドを作成する際には、コマンドの汎用性・機能性を高めるために様々なオプションを導入するが、オプションの数が増えてゆくにつれ、オプションの意味が分かりにくくなり、解析も困難になってくる。これらの問題を解決する一つの方法として、オプションおよび引数の形式から自動的にコマンドライン・パラメータ解析ルーチンを生成することが考えられる。

本研究では、オプション・引数の型，変数名との対応，省略時の暗黙値，各パラメータに対する解説文から、コマンドラインを解析し変数への値の代入を行なうルーチン，使用方法およびヘルプ・メッセージを表示するルーチンを自動的に生成するプログラムを作成した。

2．コマンドライン・パラメータ

コマンドライン・パラメータをオプション，引数，そしてキーワードに分類する。例えば、データ群をパターンと照合し、照合結果によって置換パターンを置き換え、表示・実行するコマンド match の文法は、

match <データ群> with <パターン> <置換パターン>

であり、実際には、

match x.c y.c with "%x.%y" -x "mv %x.%y %x.C"

のように指定するが、引数（<データ群>）の切れ目を示すのがキーワード（with），標準以外の使用法（<置換群>の表示ではなく実行）を指示するのがオプション（-x），その他が引数である。

オプション・引数は、対応するCプログラムの変数の型に対応して以下のように分類される。

- フラグ型　　(%b)：char型（オプションのみ）
- １０進整数型(%d)：int型
- ８進整数型(%o)：int型
- １６進整数型(%x)：int型
- 文字型　　(%c)：char型
- 文字列型　　(%s)：char　*型
- 文字列群型　(%v)：char　**型（引数のみ）
- 列挙型　　(%e)：int型

さらに、倍精度修飾（long），符号無し修飾（unsigned）も指定可能である。

3．OPTION

コマンドライン記述ファイルからコマンドライン解析ルーチンを自動的に生成するプログラムをOPTIONと呼ぶことにする。

OPTION <コマンドライン記述ファイル>

により自動的に解析ルーチンを作成する。

3．1　コマンドライン記述ファイル

コマンドラインの記述は、

Version:
　　<バージョン記述文>
Function: <機能記述文>
　　<オプション・引数・キーワード記述行>
　　<パラメータ説明文>

とする。ただし、

<オプション記述行> ::=
　　<オプション> <型> (<変数名>)　(<暗黙値>)
<引数記述行>::=
　　<型> (<変数名>)　(<暗黙値>)
<キーワード記述行> ::=
　　<キーワード>

である。 match コマンド記述を含むファイル（match.c）の一部を以下に示す。

```
/*
Version:
        第 1.00 版, 88/12/10, by 高濱徹行
Function: パターン照合を行い、照合結果を表示・実行する
        l islist
                照合結果を表示する
        x isexec
                照合結果を実行する
        %v datav
                照合を行なうデータ群を指定する
        with
        %s pattern
                照合パターンを指定する（変数は[*!?]<一文字>で指定する）
        %v result
                表示あるいは実行するパターンを指定する
*/
#include <stdio.h>
#include <match.opt>
```

3．2　データファイル

OPTIONは、言語独立性・拡張性を考慮し、エラーメッセージや生成されるルーチン中のメッセージ・パターンをデータファイルから読み込むとともに、各段階の処理

OPTION: Program for Generating Command Line Analysis Routines Automatically

Tetsuyuki TAKAHAMA

Fukui University

を関数テーブルの参照によって行なうように構成されている。以下にデータファイルの一覧を示す。

・メッセージデータファイル（option.msg）
　ＯＰＴＩＯＮのエラー・ワーニング・情報メッセージ
・解析データファイル（option.dat）
　コマンドライン記述の解析用データ
・生成ソースファイル（option.src）
　コマンドライン解析ルーチンの生成用データ

３．３　処理の流れ

ＯＰＴＩＯＮの処理は、２つの段階、すなわち、解析段階と生成段階に分けられる。

（１）解析段階

・メッセージファイルおよび解析ファイルを読み込む。
・Version, Function 等の記述項目別にそれぞれの処理関数を呼び出す。
・Version 項目では、バージョン記述文を記憶する。
・Function 項目では、機能記述文を記憶するとともに、パラメータ記述行を解析し、その結果得られたパラメータの型，対応する変数名，暗黙値，パラメータ説明文の組を記憶する。

（２）生成段階

・生成ソースファイルを読み込む。
・生成ソースファイル中に指定された処理関数を呼び出し、以下のような順でＣプログラムを出力する。
　ａ．バージョン記述文を表示する関数 version
　ｂ．パラメータ情報に基づき、コマンドの使用法を表示する関数 usage
　ｃ．変数名・型・暗黙値に基づき、変数宣言文
　ｄ．パラメータ解説文および暗黙値に基づき、ヘルプメッセージを表示する関数 help
　ｅ．オプションや引数の値を対応する変数の型に変換する型変換用サブ関数
　ｆ．コマンドラインを解析する関数 option
・メインファイルに必要な情報を以下のように出力する。
　ａ．生成されたプログラムをインクルードして使用する場合には、include 文
　ｂ．外部ファイルとして使用する場合には、全ての変数の外部変数宣言（extern）文
　ｃ．関数 main 定義部と関数 option 呼び出し部

右に match コマンド記述から生成されたルーチン群を含むインクルードファイル（match.opt）を示す。

４．終わりに

ＯＰＴＩＯＮコマンドの使用によって、
　ａ．コマンド制作者はコーディング量が少なくなり、コマンドライン記述の付加によってプログラムが分かりやすくなる。
　ｂ．コマンド利用者は統一された形式での使用法およびヘルプメッセージを得ることができる。

という利点がある。

今後、より豊富な変数の型およびコマンドラインの形式を提供するようにＯＰＴＩＯＮを改良して行く予定である。

```
/*----- Functions for option analysis -----*/

static void version(command)
char *command;
{
        fprintf(stderr, "%s: 第 1.00 版, 88/12/10, by 高橋徹行¥n", command);
        exit(0);
}

static void usage(command)
char *command;
{
        fprintf(stderr, "使用法: %s [-hlx] %%v with %%s %%v¥n", command);
        exit(1);
}

static char O_FLG_1;
#define islist  (O_FLG_1 & 0x1)
#define isexec  (O_FLG_1 & 0x2)
static char **datav;
static char *pattern;
static char **result;

static void help(_command)
char *_command;
{
        printf("***** %s: パターン照合を行い、照合結果を表示・実行する *****¥n", _command);
        printf("¥t-h: ヘルプ・メッセージ¥n");
        printf("¥t-l: 照合結果を表示する¥n");
        printf("¥t-x: 照合結果を実行する¥n");
        printf(" <配列>: 照合を行なうデータ群を指定する¥n");
        printf(" with¥n");
        printf(" <文字列>: 照合パターンを指定する（変数は[*!?]<一文字>で指定する）¥n");
        printf(" <配列>: 表示あるいは実行するパターンを指定する¥n");
        exit(0);
}

/* Boolean Switch */

/* String */

static void option(argc, argv)
int argc; char *argv[];         /* NULL is end of args */
{
        int args=0, c, i; char *options;

        for(i=1; i<argc; ++i)
            if(*(options=argv[i])=='-') {
                while((c=(*++options))!=NULL)
                    switch(c) {
                    case 'l':
                        O_FLG_1|=0x1;   break;
                    case 'x':
                        O_FLG_1|=0x2;   break;
                    case 'h':
                        help(argv[0]);
                    default:
                        goto error;
                    }
            }
            else
                switch(++args) {
                case 1:
                        datav=&argv[i];
                        while(i<argc)
                                if(strcmp(argv[i], "with")==0) break;
                                else ++i;
                        if(i==argc) goto argerror;
                /*  2: */
                             ++args;
                        argv[i]=NULL;
                        break;
                case 3:
                        if((pattern=argv[i])==NULL) goto error;
                        break;
                case 4:
                        result=&argv[i];
                        return;
                default:
                        goto error;
                }
        if(args==4) return;
    argerror:
    error:
        usage(argv[0]);
}
```