gccのエラーメッセージの日本語化

滋賀職業能力開発短期大学校 情報処理科
2 回生 古市 多華子
講師 高岡 正

1994年3月4日

概要

C 言語で書いたプログラムは、機械が実行できる言語に翻訳しなければならない。この翻訳プログラムを C コンパイラという。gcc[1]は C コンパイラの一つである。ソースプログラム中の誤りを検出し、エラーメッセージを出すことはコンパイラの重要な機能である。プログラムを書く人はエラーメッセージを参考にして誤りを修正していく。C 言語のソースプログラム中に誤りがあって、機械語に翻訳できない時、gcc は英語のエラーメッセージを出す。この研究では、gcc が出す英語のエラーメッセージを日本語のエラーメッセージに変換して出力するプログラムを作成した。

1 はじめに

gcc はGNU のC コンパイラで、Richard Stallman氏によって開発されたフリーソフトウェアである。プログラムが一般公開されているので、その内容を見ることはもちろん、変更することもできる。

gccで誤りのあるソースプログラムをコンパイルすると、エラーメッセージが出される(図 1)。

プログラミングを始めてからあまり時間のたっていない人にとって、このエラーメッセージは理解しにくく、初めから読もうとしない場合もある。そこで、エラーメッセージが日本語で出れば、誤りの修正がしやすくなり、C言語でプログラムを書く人の役に立つと考えた。

このプログラムをtccということにする。オリジナルの gcc を gcc-original という名前に変更し、tcc の別名を gcc にする(図 2)。そうして gcc を起動した時に tcc が起動されるようにした。

tcc を起動すると、その内部でオリジナルの gcc が起動される。オリジナルの gcc から出されたエラーメッセージをtccに読み込む。データベースを使って日本語に変換し、日本語のエラーメッセージを出力する(図 3)。

tcc の記述言語には perl[2]というプログラミング言語を選んだ。

2 実現

2.1 プログラムの概要

エラーメッセージを日本語化する時に、gccのプログラム中のエラーメッセージを直接置き換える方法がある。しかし、gcc は規模が大きく、置き換える作業に大変な時間がかかる。また、バージョンアップが頻繁に行われるので、特定のバージョンでしか使えなくなってしまう。

そこで、gcc のプログラムには手を加えずに、別のプログラムの内部でgccを起動して、出てきたエラーメッセージを日本語に置き換えることにした。

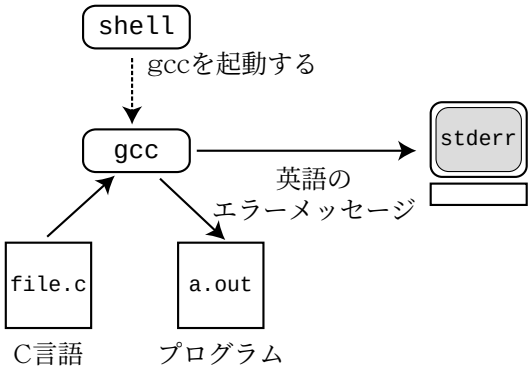

図1: gccを起動した時

```
kterm
tf@ws02:/usr/local/bin$>ls -l gcc*
lrwxrwxrwx  1 root     t2            22 Aug 26  1993 gcc -> /home/fs/92/17/bin/tcc*
-rwxr-xr-x  1 t2       wheel     114688 Aug 26  1993 gcc-2.4.3.1*
-rwxr-xr-x  2 t2       wheel     114688 Feb 22 09:18 gcc-2.5.8*
lrwxrwxrwx  1 t2       wheel          9 Feb 22 15:01 gcc-original -> gcc-2.5.8*
tf@ws02:/usr/local/bin$>
```

図2: リンク

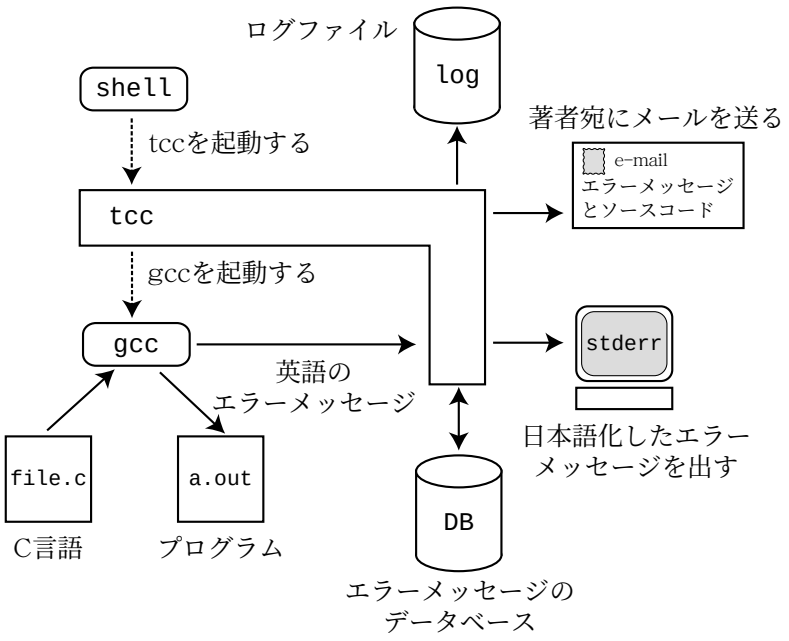

図3: tccを起動した時

2.2 記述言語

perlはLarry Wall氏によって開発されたプログラミング言語である。移植性の高いフリーソフトウェアで、さまざまな環境で使えるようにすることが簡単にできる。

また、インタプリタ言語なので、記述や修正をしてからすぐに実行することができる。プログラムの開発にかかる時間はC言語のようなコンパイル言語より短くて済む。

perlには次のような特長もある。

- C言語とよく似た構文で、フリーフォーマットである。
- シェルやC言語などから多くの機能を受け継いでいる。
- パターンマッチ、置換、文字列の連結などの文字列処理が簡単に扱える。
- 任意の文字列を添字にできる連想配列が使える。
- UNIXシステムコールやネットワーク操作、DBMによるデータベース操作などが関数で用意されている。

どのような環境でも使用できるプログラムを作り、プログラミングを短期間で行うためにperlをプログラムの記述言語に選んだ。

2.3 実現の過程

2.3.1 パイプを使う

はじめは、1つのプロセスでgccを起動し、標準エラー出力に出されたエラーメッセージをパイプ[3]を通して読み込むようにした。gccを起動して標準エラー出力をオープンし、gccというファイルハンドルと結びつける。

これで、標準エラー出力に出されたエラーメッセージはgccというファイルハンドルから読み込めるようになった(図4)。

```
# gcc を起動する。
open(GCC, "$gcc @ARGV 2>&1 |") || die "cannot invoke gcc";
while (<GCC>) {
        s/$gcc/gcc/;
        print;
        $error[$line++] = "$_";
}
close GCC;
```

図4: はじめのプログラム

```
emacs @ ws02
#include <stdio.h>

main()
{
    printf(MESSAGE);
}

 [----]-----NEmacs: sampl.c            (EJE-:C)--All----------------
cd /tmp_mnt/home/fs/92/17/soturon/
test2.perl -DMESSAGE='"Test program\n"' sampl.c
sampl.c: 関数'main'において:
sampl.c:5: 'Test'が宣言されていません(この関数内部で初めて使われました)
sampl.c:5: (宣言されていない変数や関数については1度だけ報告されます.
sampl.c:5: 各関数内で見つけられることに報告されません)
sampl.c:5: 'program'の前に構文上の間違いがあります
sampl.c:5: エスケープ文字'\'が'や"で囲まれていないところで使用されました

Compilation exited abnormally with code 1 at Mon Mar  7 19:59:38

 [----]--**-NEmacs: *compilation*        (-JE-:Compilation: exit)--All----
(No files need saving)
```

図5: はじめのプログラムを実行した時のエラー

2.3.2 問題点

パイプによってエラーメッセージを読み込めるようになったが、起動する時の引数の渡し方に問題があることがわかった。

1つのプロセスで実行する場合、tcc を起動した時にシェルに引数が解釈される。そして、tcc 内部でgccを起動した時にもう一度引数が解釈される。

たとえば、gccを起動する時に引数でマクロを定義する場合、置換する文字列に空白が含まれることがある。この引数は二度解釈されることによって一つのまとまった文字列であると判断されない。これでは、目的プログラムを作成できなくなってしまう(図5)。

2.3.3 プロセスを fork する

この問題を解決するために、シェルにとって意味のある文字の前にバックスラッシュを挿入し、シェルが解釈しないようにしてみた。しかし、この方法ではemacsの中からgccを起動した時に、バックスラッシュの入ったコンパイルコマンドがエコーされてしまい、見苦しい。

そこで、プロセスをフォークし(図6)、子プロセスで gcc を起動して引数がそのまま渡されるようにした。

まず、パイプを作る。続いて fork システムコールを実行する。親プロセス側は子プロセスのプロセス id を返す。親プロセス側でパイプの書き込み端を閉じる。

fork システムコールを実行すると、子プロセス側は0を返す。子プロセスであれば、パイプの読み込み端を閉じる。子プロセスで gcc を起動するので、子プロセスの標準エラー出力と親プロセス側のパイプの読み込み端をつなぐ(図7)。

パイプがつながったら子プロセスgccとなる。子プロセス側で gcc を起動するときに、perl の exec 関数を使う。perl の exec 関数は C ライブラリの execvp(3)を実行する。引数にメタキャラクタを含んでいない時は、効率を考えてシェルに渡さずに、直接execvp(3)に引数を渡す。これで、引数は二度解釈されなくなる。

```
#
#  gccを起動する。
#
pipe(RD, WR);                   # gccとの通信用のパイプをオープンする。
if (($pid = fork) > 0) {        # 親プロセス側。
    close WR;                   # tcc側でパイプの書き込み端を閉じる。
} elsif ($pid == 0) {           # 子プロセス側。
    close RD;                   # パイプの読み込み端を閉じる。
    open(STDERR, ">&WR");       # 標準エラー出力とtcc側のパイプの書き込み端をつなぐ。
    close WR;
    exec $gcc, @ARGV;           # gccを起動する。
    die "$gcc がありません。"; # gccが起動できなかった時。
} else {                        # フォークできなかった時。
    die "フォークできません: $!\n";
}

while (<RD>) {                  # エラーメッセージを読み込む。
    s/$original/gcc/;
    $error[$line++] = "$_";
}
close RD;
```

図6: forkを実行した時の流れとプログラム

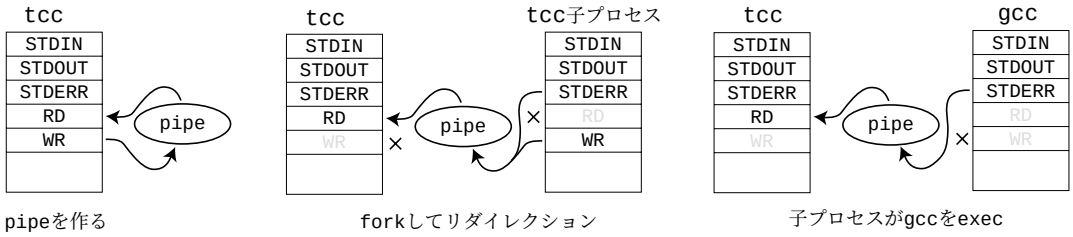

図7: パイプの生成とforkによるパイプの連結

3　日本語への変換

エラーメッセージには、どこでエラーが起こったのかがわかるような情報が付けられている。

tccの内部に読み込んだエラーメッセージを日本語に変換して出力する時、変換する前のメッセージの形と同じ形にする必要がある。これは、emacsの中でgccを起動した時、エラー行へのジャンプ機能を使えるようにするためである。

3.1　エラーメッセージの解析

ソースプログラム sample.c の 20 行目で宣言していない変数iを使用していたとすると、次のような形でメッセージが出される。

```
sample.c:20: `i' undeclared (first use
this function)
```

この中で、エラーメッセージは

```
`i' undeclared (first use this function)
```

である。

関数 function に渡す引数に誤りがあるとすると、

```
sample.c:30: warnig: passing arg 1 of
`function' from incompatible pointer type
```

という警告メッセージが出される。この中で、エラーメッセージは

```
passing arg 1 of `function' from
incompatible pointer type
```

である。

上の形になっていないメッセージがある。定義していない関数print を使用した時には、次のようなメッセージが出される。

```
ld: Undefined symbol
     _print
```

未定義のシンボル名は、_ をつけて出されるが、これはそのまま出すようにした。

3.2　パターン化の考え方

3.2.1 英語のエラーメッセージのパターン化

エラーメッセージには変化する部分と共通の部分がある。たとえば、

```
`i' undeclared
`j' undeclared
```

のように、` 'の中は、ソースプログラムごとに違う値が入るが、それ以外は共通というエラーメッセージがある。

これらを異なるエラーメッセージとみなすと、データベース化できない。そこで、変化する部分をパターンに置き換え、同じエラーメッセージとして扱うことにした。

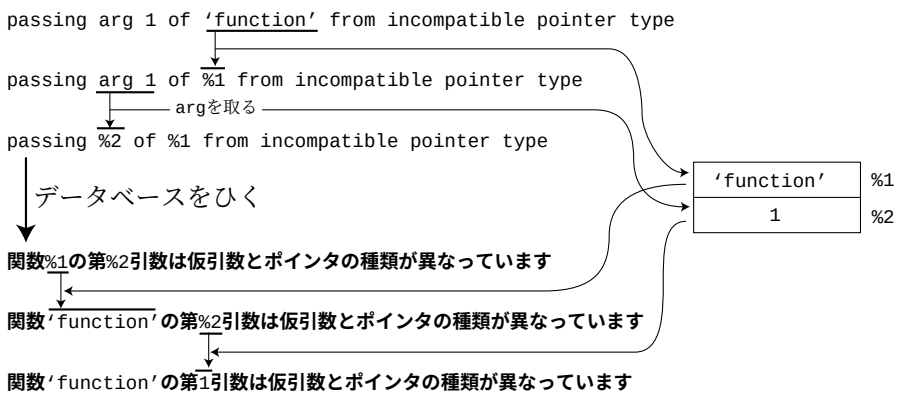

図8: エラーメッセージの変換

まず、エラーメッセージを見て、変化する部分のパターンマッチを行う。パターンにマッチした部分の前後をそのまま取り出し、マッチした部分を"%数字"に置き換える。マッチした部分(変化する値)は、配列に保存しておく。この操作を、変化する部分がなくなるまで繰り返す。

上のエラーメッセージは

```
%1 undeclared
```

という形になる。

エラーメッセージの変化する部分は、`'で囲まれた変数名や関数名のほかに単項演算子や引数の番号、二項演算子、文字コードがある。

- `invalid type argument of `unary *'`
- `type of formal parameter 1 is incomplete`
- `invalid operands to binary *`
- `parse error before character 0262`

上のエラーメッセージは、下線を引いた部分が変化する。

単項演算子に関する１つ目のエラーメッセージは`'で囲まれている中でunaryの後ろだけが変化する。変数名や関数名の処理は、`'!部分をそのまま保存するので、これとは別にパターンマッチを行うようにした。

引数の番号の前にはarg、 argument、parameterがついている。パターンマッチの時には、arg 数字、argument 数字 、parameter 数字にマッチするものを％数字に置き換える。保存する値は引数の番号だけにし、日本語に変換する時にarg、argument、parameterがそのまま入らないようにした。

3.2.2 日本語のエラーメッセージへの変換

パターン化したエラーメッセージを日本語に変換する時に、データベースを使っている。

```
%1 undeclared
```

をキーにすると、日本語のパターン

%1 が宣言されていません

を引くことができる。

日本語のエラーメッセージを出す時には、保存しておいた値を元に戻さなければならない。エラーメッセージの中に％数字になっているところがあれば、配列に保存した値と置き換える。

配列に`i'が保存してあれば、%1を`i'に置き換えて、

`i' が宣言されていません

というエラーメッセージを出す。

図８に英語のエラーメッセージをパターン化して、日本語のエラーメッセージに変換する様子を示す。

3.3 データベース化

エラーメッセージを変換する時に使っているデータベースファイルは

英語のパターン <TAB> 日本語のパターン

という書式にしている。

英語のエラーメッセージに対応する日本語のエラーメッセージをデータベースファイルから探す時に、線形サーチを使わないで、連想配列を利用することにした。

連想配列は値の対をリストにしたもので、値の対の1番目のものはキーと呼ばれ、添字に用いられ

```
emacs @ ws02
Date: Mon, 07 Mar 1994 23:02:14 JST
To: tf@cs.shiga-pc.ac.jp
From: Takako Furuichi <tf>
Subject: tcc log

user: tf
time: 94 3 7, 23:2:12
args: gcc -o sample1 sample1.c

sample1.c: In function 'main':
sample1.c:11: conflicting types for 'a'
sample1.c:10: previous declaration of 'a'
sample1.c:16: subscripted value is neither array nor pointer
***new error pattern***
conflicting types for %1
===== sample1.c =====
1       #include <stdio.h>
2
3       #define MAX 20
4       #define I   12
5       #define J   12
6       #define M   4
7
8       void main()
9       {
10          double a[MAX][MAX], d=1.0;
11          int i, j, a;
12          void matprn();
13
14          for (i = 1; i <= I; i++) {
15              for (j = 1; j <= J; j++) {
16                      a[i][j] = 0.0;
17              }
18          }
19
20          matprn(a, I, J);
 [----]-----N(show-+inbox)  +inbox/830     (-JE-:Fundamental)--Top----------
inc +inbox...done
```

図9: メールで送られたエラーメッセージ

る。キーは、そのハッシュ値が計算されてハッシュ表に登録されている。それで、登録しているリストの個数に関係なく高速に検索を行うことができる。

英語のエラーメッセージをキーにして、日本語のエラーメッセージを取り出すために、データベースファイルを連想配列と結び付けなければならない。DBM ファイルを連想配列に結合する dbmopen 関数を使ってデータファイルを処理する。

これで、パターン化した英語のエラーメッセージを添字にして日本語のエラーメッセージを参照することができるようになる。

4　エラーメッセージの収集

gccにどのようなエラーメッセージがあるかを知るために、コンパイルした時に実際に出されたエラーメッセージを電子メールを使って著者宛に送ることにした(図9)。また、どのような誤りがある時に出されたエラーメッセージなのかがわかるように、ソースプログラムも送るようにした。そして、送られて来たエラーメッセージから順にデータベースファイルに登録していった。

最初はエラーメッセージが出たソースプログラムをすべて送るようにしていた。多い時は1日で200通以上のメールが来たこともあった。このメールの処理が大変なので、データベースに登録していないエラーメッセージがある時だけメールを送るように変更した。

データベースに登録しているエラーメッセージは210種類ある。バージョン2.4.3のプログラム中にいくつのエラーメッセージがあるのか調べると、約1300種類あり、そのうち警告メッセージは300だった。

研究をしている間にgccのインストールが2回行われた。始めた時がバージョン2.3.1で、2.4.3に変わり、現在は2.5.8である。バージョン2.4.3から2.5.8になって出力のされ方が変化したメッセージがある。

図10: 実行結果

5　考察

5.1　結果

オリジナルのgccの実行結果とtccの実行結果を図 11 に示す。

オリジナルのgccを起動した時と、tccを起動した時の実行時間にどれくらいの差があるのかを調べた。約1100行の誤りのないソースプログラムを20回コンパイルし、実行時間をtimeコマンドで計測する。平均を求めた結果は、次のようになった。

	実時間	ユーザ	システム
gcc–original	25.7 秒	9.4 秒	3.3 秒
tcc	25.9 秒	0.0 秒	0.1 秒

次に、誤りがあった場合について実行時間を計測した。5行のエラーメッセージが出た場合、10行のエラーメッセージが出た場合、20 行のエラーメッセージが出た場合の結果は次のようになった。

	実時間	ユーザ	システム
gcc–original	0.8 秒	0.1 秒	0.2 秒
tcc	1.1 秒	0.1 秒	0.1 秒

	実時間	ユーザ	システム
gcc–original	0.9 秒	0.2 秒	0.3 秒
tcc	1.4 秒	0.2 秒	0.1 秒

	実時間	ユーザ	システム
gcc–original	0.9 秒	0.1 秒	0.3 秒
tcc	1.7 秒	0.3 秒	0.2 秒

実時間はコンパイル終了まで感じている時間、ユーザはユーザーモードで消費したCPU時間、システムはシステムモードで消費したCPU時間である。tccのユーザとシステムには、オリジナルのgccが消費した時間が含まれていない。

正確な時間を計測することはできないが、この結果から誤りがなかった時は、tccを実行したことによるオーバーヘッドはほとんど気にすることがないと考える。

しかし、エラーメッセージが出された時はエラーの行数が増すほど、オリジナルのgccとtccの実時間の差が大きくなることがわかる。

5.2　ログファイルについて

1993 年 11 月から gcc を起動した時の情報をログファイルに書き込むよ

うにした。情報は、

- ログイン名
- 起動時刻(年月日、時刻)
- gcc の引数
- エラーメッセージを出力する時の言語
- エラーメッセージがあるときはエラーメッセージ

の５つである。

このファイルを使って、どのようなエラーメッセージが多く出されているのか、起動回数はどれくらいなのか、1年生と2年生ではエラーの内容に違いがあるのかを調べてみた。

情報処理科の１年生と２年生を対象に、エラーメッセージの種類と頻度を出力するプログラムを perl で作成し、ログファイルを処理した。

エラーメッセージで、最もよく出されているのは、

- `parse error before •••:`
  •••の前に構文上の間違いがあります
- `••• undeclared (first use this function):`
  •••が宣言されていません(この関数内部で初めて使われました)

であることがわかった。

ソースプログラムを見ると、

- 宣言していない変数を使用する
- セミコロンがない
- かっこの対応がとれていない

という誤りが多いことがわかる。

また、変数名が重なってしまったり、マクロ定義をし忘れた識別子を使っ

ているという誤りもあった。その他、

- 関数名の記述を間違えている
- gcc の引数でライブラリの指定を忘れている

という誤りから出されたエラーメッセージが多かった。

プログラムの内容でエラーメッセージの種類が変化することもわかる。数値計算や画像処理では配列を使ってプログラムを書くので、配列に関するエラーメッセージが多かった。

1、２年生とも、

- `subscripted value is neither array nor pointer:`
  添字がつけられている変数は配列でもポインタでもありません

が多く、２年生は、

- `invalid use of array with unspecified bounds:`
  大きさを指定していない配列が使われていますが無効です

１年生は、

- `array subscripted is not an integer:`
  配列の添字が整数でありません

が多かった。

個人別の gcc 起動回数を表にする。総起動回数は、情報処理科の１年生と２年生の合計である。

年月	個人最多	個人最少	総起動回数
93.11	648	7	8151
93.12	956	3	8597
94.1	579	2	4747
94.2	816	7	8002

gccの起動回数は個人差が大きいので、あるユーザが出したエラーが多くなっていることもある。

5.3 問題点

次の問題点がある。

- エラーメッセージは、コンパイルした時に出されたものだけを日本語化しているので、すべてのエラーメッセージを日本語で出すことができない。

これは、gccのプログラムからエラーメッセージだけを抜き出して、対応する日本語のエラーメッセージを考えればいいのかもしれない。しかし、私にとって英語のエラーメッセージだけを見て日本語に訳すことが難しい時があった。

現在は、電子メールで未登録のエラーメッセージが送られてきたら、できるだけ早く日本語訳をつけて登録するようにしている。

- データベースに登録していないエラーメッセージは、電子メールで著者に送り、日本語訳とともに追加登録しているが、これを自動化できない。

英語のエラーメッセージをパターン化したものをデータベースに登録することはできる。この問題は、翻訳機を使用したとしても適切な日本語の表現かどうかをチェックする必要があるため、解決は難しい。

翻訳機にかけるとすると、まず英語のエラーメッセージを英語の文法に合うように変換しなければならないと思う。また、C言語やコンピュータ用語の辞書が必要になるだろう。

- ログファイルのサイズが非常に大きくなってしまう。

1ヵ月で2メガバイト近くになることもある。ディスクの容量には限りがあるので、あまり大きくなるとユーザみんなに迷惑をかけてしまう。現在は1カ月ごとにまとめて圧縮するようにしている。

エラーメッセージの頻度だけを調べるのならば、データベースを使ってエラーメッセージと頻度を登録しておけば、小さくまとめることができる。

- エラーメッセージの出力の形が変化すると、プログラムを修正しなければならなくなってしまう。

Include ファイル内でのエラーをあらわすメッセージ、"In file included from ファイル名:行番号 , from ファイル名:行番号･･･" は、バージョン 2.4.3 から 2.5.8 になって、出力のされ方が変化している。バージョン2.4.3では1行で出されていたが、バージョン 2.5.8 は from ごとに改行して出されている。

これは、1番最後の from 以下にあるファイルがネストの1番外側なので、日本語で出力する時は、このファイル名から逆順に出力していくことになる。エラーメッセージを1行読み込んで日本語に変換し、出力する現在の方法では対応することができないので、プログラムの構造を考え直す必要がある。

5.4 課題

次のことを今後の課題として考えている。

- 日本語エラーメッセージの中には直訳に近いものもあるので、わかりやすいものに変えていく。
- 誤りの修正を助ける機能を付け加える。

英語のエラーメッセージを日本語に変える時に、C言語の図書[4]、[5]などを参考にして考えてきた。日本語にするとわかりにくくなってしまう言葉もあったが、直訳で出している。

エラーメッセージは、簡潔で、しかも意味がわかるようにしなければならない。しかし、英語を直訳したメッセージを見て、誤りがあることはわかっても、どこを修正すればいいのかがわからないこともある。そこで、考えられる誤りをメッセージとして出力すれば、修正がしやすくなるだろう。

- 日本語のエラーメッセージと英語のエラーメッセージを同時に出せるよう

現在は、日本語のエラーメッセージを出力するために、環境変数を参照している。値が代入してあれば、英語のエラーメッセージをすべて日本語に変換して出力している。日本語のエラーメッセージだけを見ていると、オリジナルのメッセージが理解できなくなるかもしれない。

これを、英語のエラーメッセージを出力してから、スイッチで日本語に切替えたり、同時に出力するようにすれば、日本語が使えない環境で英語のエラーメッセージを見た時に、その内容がわかるようになると思う。

- エラーメッセージの登録数を増し、データベースを充実する。

これは、C言語のさまざまな機能を使ったプログラムを書いて、エラーメッセージを出すようにすることが必要である。

しかし、C言語やコンパイラについての知識が豊富でないと、期待したエラーメッセージを出せないことが、サンプルプログラムの作成を通してわかった。

- 日本語だけでなく、多国語に対応できるプログラムへ拡張する。

データベースを作成するのが大変だが、プログラム本体は日本語変換と同じものを使うことができる。

- ログファイルを有効に活用する。

ログファイルの記録から、ある個人について時間の経過とエラーの内容の変化を調べることができる。また、同じファイルに対してどのように目的プログラムを完成させていったかを知ることもできる。

このようなログファイルを元にして作成したデータを活用して、C言語の学習過程における教育の効果をみることができる。意味のあるデータを作成するために、ログファイルに記録する情報や、フォーマットをさらに検討したい。

5.5 フリーソフトウェアについて

この研究で使ってきたgcc もperl もフリーソフトウェアである。これらの優れたソフトウェアを無料で手に入れ、使用できることは、すばらしいことだと思う。開発したソフトウェアを公開し、自由に使ってもらうことで、自分では気づかなかった問題点を教えてもらうことができる。その点を改善していけば、さらに使いやすいソフトウェアになっていくと思う。

謝辞

情報処理科の先生方には、協力をしていただいて感謝している。作成したプログラムを実習で使ってくれた情報処理科の学生に感謝する。

参考文献

[1] gcc(1)--gcc オンラインマニュアル

[2] perlプログラミング, Larry Wall and Randal L. Schwartz 著 , 近藤嘉雪 訳 , SOFTBANK

[3] UNIX システムコール・プログラミング , Marc J. Rochkind 著 , 福崎俊博 訳 , ASCII

[4] ANSI C言語大辞典, マーク・ウィリアムズ社編, 坂本文, 今泉貴史 監訳, パーソナルメディア株式会社

[5] 新C 言語入門 - シニア編 , 林晴比古 著 , SOFTBANK

A　ログファイルから作成したデータ

gcc は 1 年生 26 人、2 年生 14 人が使っている。

A.1　エラーメッセージとその頻度

下の表は 1993 年 11 月から 1994 年 2 月のログファイルに現われたエラーメッセージで最も多かったものから順にあげている。かっこ内の数字は各学年の頻度の順位を示す。

エラーメッセージのパターン	全体	1 年生		2 年生	
%1 undeclared (first use this function)	6913	4166	(2)	2747	(1)
parse error before %1	5861	4381	(1)	1480	(2)
data definition has no type or storage class	2130	1188	(3)	942	(3)
%1 undeclared here (not in a function)	1312	683	(6)	629	(5)
coflicitng types for %1	1281	555	(9)	726	(4)
subscripted value is neither array nor pointer	1149	720	(5)	429	(8)
Undefined symbol	1066	854	(4)		
type mismatch with previos external decl	824			460	(6)
type mismatch with previous implicit declaration	705			365	(10)
No such file or directory	702	601	(8)		
%1 was previously implicitly declared to return %2!	683				
array subscript is not an integer	646	534	(10)		
execess elements in scalar initializer after %1	611	611	(7)		
passing %2! of %1 from incompatible pointer type	610				
declaration for parameter %1 but no such parameter	498			455	(7)
invalid use of array with unspecified bounds	418			407	(9)

A.2　gcc の起動回数

1993 年 11 月から 1994 年 2 月の gcc の起動回数を下の表に示す。

	合計	1 年生	2 年生
起動総数	29497	22969	6528
1 人当たり	756	919	466

A.3　エラー率

このデータは情報処理科の学生以外のデータを含んでいる。

年月	起動回数	全体のエラー率(%)	日本語(%)	英語(%)
1993 年 11 月	8644	30.7	31.3	68.7
1993 年 12 月	8957	24.6	40.6	59.4
1994 年 1 月	5428	31.0	37.1	62.9
1994 年 2 月	8690	27.6	63.7	36.3

B プログラムリストと解説

```
 1 #! /usr/local/bin/perl
 2 #
 3 #    $Id: tcc,v 1.10 1993/12/21 06:19:40 tf Exp tf $
 4
 5 #  $gcc          : オリジナル gcc のパス名
 6 #  $databasefile : データベースのパス名
 7 #  $logfile      : gcc 起動の情報を書く log ファイルのパス名
 8 #  $errorreport  : エラーメッセージを送るアドレス
 9 #  $language     : エラーメッセージを出力する言語
10
11 #
12 #  Configuration
13 #
14 $gcc = "/usr/local/bin/gcc-original";
15 $databasefile = "/home/fs/92/17/soturon/data";
16 $logfile = "/home/fs/92/17/soturon/tcclog";
17 $errorreport = "tf@cs.shiga-pc.ac.jp";
18 $language = "original";
19
20 # gcc を起動した時刻を得る。
21 ($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) = localtime(time);
22 $date = sprintf("%4d-%02d-%02d %02d:%02d:%02d", $year + 1900, ++$mon, $mday,
    $hour, $min, $sec);
23 # gcc を起動したユーザの情報を得る。
24 ($name, $passwd, $uid, $gid, $quota, $comment, $gcos, $dir, $shell) = getpwuid($<);
25 @tmp = split(/\//, $gcc);
26 $original = pop(@tmp);
27
28 # 環境変数で日本語指定した場合の処理
29 $language = $ENV{'GCCERROR'} if defined($ENV{'GCCERROR'});
30
31 # 引数で指定した場合の処理。
32 foreach $argnum (0 .. $#ARGV) {
33     if ($ARGV[$argnum] =~ /\-language/) {
34         $language = $ARGV[$argnum+1];
35         splice(@ARGV, $argnum, 2);
36         last;
37     }
38 }
39
40 #
41 #  gcc を起動する。
42 #
43 pipe(RD, WR);                # gcc との通信用のパイプをオープンする。
44 if (($pid = fork) > 0) {     # 親プロセス側。
45     close WR;                # tcc 側でパイプの書き込み端を閉じる。
46 } elsif ($pid == 0) {        # 子プロセス側。
47     close RD;                # パイプの読み込み端を閉じる。
48     open(STDERR, ">&WR");    # 標準エラー出力と tcc 側のパイプの書き込み端をつなぐ。
49     close WR;
50     exec $gcc, @ARGV;           # gcc を起動する。
51     die "$gcc がありません。";  # gcc が起動できなかった時。
52 } else {                        # フォークできなかった時。
53     die " フォークできません: $!\n";
54 }
55
56 while (<RD>) {                  # エラーメッセージを読み込む。
57     s/$original/gcc/;
58     $error[$line++] = "$_";
59 }
60 close RD;
61
62 # logfile に情報を書き込む。
63 if (defined($logfile)) {
64     open(LOG, ">>$logfile");
65     print LOG "$date $language\t$name\tgcc @ARGV\n";
```

```
 66     for ($i = 0; $i < $line; $i++) {
 67         print LOG "\t$error[$i]";
 68     }
 69     print LOG "\n";
 70     close LOG;
 71 }
 72
 73 # エラーのない時は、0 を返す。
 74 exit(0) if ($line == 0);
 75
 76 #
 77 # 日本語変換の処理を行う。
 78 #
 79 dbmopen(%ERR, "$databasefile", undef);  # データベースファイルをオープンする。
 80 select(STDERR);
 81 # 各エラーメッセージを処理する。
 82 foreach $mesg (@error) {
 83
 84     if ($mesg =~ /^\s+_\w+|\S+ returned [0-9] exit status/) {
 85         print "$mesg";
 86         next;
 87     } elsif ($mesg =~ /^In file included from|^\s+from\S+|
                                        ^Reading specs from/) {
 88         print "$mesg";
 89         next;
 90     }
 91
 92 #   メッセージをコロンで分割し、エラーの内容を取り出す。
 93     chop($mesg);
 94     @fields = split(/:/, $mesg); # ':' で行を区切ってみる。
 95
 96     if (@fields == 1) {          # ':' がなかった場合。
 97         $position = "";
 98         $errmsg = $fields[0];
 99     } elsif (@fields >= 3 && $fields[1] =~ /^[0-9]+$/) {
                                     # ':' が 2 つ以上あって、2 つ目が行番号の時。
100         $position = shift(@fields) . ":";
101         $position .= shift(@fields) . ":";
102         $errmsg = join(':', @fields);
103     } else {                     # それ以外の時。1 つ目をファイル名だと考える。
104         $position = shift(@fields) . ":";
105         $errmsg = join(':', @fields);
106     }
107
108     $pad = ""; $errmsg =~ s/^(\s+)//; $pad = $1;
109     ($pat, @arg) = &encode($errmsg); # エラーメッセージをパターン化する。
110
111     if (!defined($ERR{"$pat"})) {# dbm ファイルに登録されていなければ、新しいエラー。
112         push(@new_error, $pat);
113         print "$position$pad$errmsg\n";
114     } elsif ($language eq "japanese") { # 日本語の設定をしている時は、日本語を出力。
115         $jp = $ERR{"$pat"};
116         ($jmsg) = &decode($jp, @arg);
117         print "$position$pad$jmsg\n";
118     } else {            # language が original または english なら英語を出力。
119         print "$position$pad$errmsg\n";
120     }
121 }
122
123 #   新しいエラーパターンがあれば、メールで送る。
124 if (defined($errorreport) && @new_error > 0) {
125     open(MAIL, "| mail -s 'tcc log' $errorreport") || die "can't open mail";
126     print MAIL "$date $language\t$name\tgcc @ARGV\n";
127     $, = "\n";
128     print MAIL "\n===== Error message =====\n";
129     print MAIL @error;
130     print MAIL "\n\n===== New error message =====\n";
131     print MAIL @new_error;
```

```
132     print MAIL "\n";
133     $, = "";
134
135     foreach $i (0 .. $#ARGV) {
136         if ($ARGV[$i] =~ /\w+\.c/) {
137             $file = $ARGV[$i];
138             open(FILE, $file) || next;
139             $l = 1;
140             print MAIL "\n===== $file =====\n";
141             while ($buf = <FILE>) {
142                 printf MAIL "%5d\t%s", $l, "$buf";
143                 $l++;
144             }
145        }
146        close FILE;
147     }
148     close MAIL;
149 }
150
151 #   エラーがあったので、1 を返す。
152 exit 1;
153
154 ##########################################################
155
156 #   パターン化したメッセージとメッセージに固有の値がはいった配列を返す。
157
158 sub encode {
159     local($msg) = @_;
160     local($num, @tmp, @str);
161
162 #   単項演算子がある時
163     if ($msg =~ /`unary ([^']+)'/) {
164         $str[++$num] = $1; $msg = "$`%$num$'";
165     }
166 #   ` ' で囲まれた値がある時
167     while ($msg =~ /`([^'`]+)'/) {
168         $str[++$num] = $1; $msg = "$`%$num$'";
169     }
170 #   ` ' で囲まれていない引数の番号がある時
171     while ($msg =~ /\b(arg|argument|parameter) ([0-9]+)/) {
172         $str[++$num] = $2; $msg = "$`$1 %$num$'";
173     }
174 #   ` ' で囲まれていない値がある時
175     if ($msg =~ /^(invalid operands to binary )([^ ]+)/) {
176         $str[++$num] = $2; $msg = "$1%$num$'";
177     } elsif ($msg =~ /^(parse error before character )(\w+)/) {
178         $str[++$num] = $2; $msg = "$1%$num$'";
179     } elsif ($msg =~ /^(gcc version )([^ ]+)/) {
180         $str[++$num] = $2; $msg = "$1%$num$'";
181     } elsif ($msg =~ /([^:]+)(: No such file or directory)/) {
182         $str[++$num] = $1; $msg = "$`%$num$2$'";
183     }
184     return ($msg, @str);
185 }
186
187 ##########################################################
188
189 #   日本語のメッセージに保存しておいた値を戻す。
190
191 sub decode {
192     local($msg, @str) = @_;
193     local($num, $x);
194
195     for ($num = 1; $num <= @str; $num++) {
196         $x = "%$num";
197         $msg =~ s/$x/$str[$num]/g;
198     }
199     return $msg;
200 }
```

C　データベースファイル

```
# This is gcc error database
# format
#   Originalmessage <TAB> Japanesemessage
# $Id: data,v 0.2 1993/11/08 05:32:38 tf Exp tf $

%1 declared %2 but never defined
        %1 は %2 と宣言されていますが定義されていません
%1 declared `static' but never defined
        %1 は static と宣言されていますが定義されていません
%1 declared as function returning a function
        %1 は関数を返す関数として定義されています
%1 declared inside parameter list
        関数に渡す引数を書くところで %1 の型の定義が行われています
%1 expects "FILENAME" or <FILENAME>
        %1 では "FILENAME" または <FILENAME> でファイル指定しなければなりません
%1 function does return
        %1 function does return
%1 has both %2 and initializer
        %1 は %2 宣言と初期化が同時に行われました
%1 initialized and declared %2
        %1 は初期化されて %2 宣言されています
%1 may be used uninitialized in this function
        %1 may be used uninitialized in this function
%1 previously declared here
        %1 は以前にここで宣言されました
%1 previously defined here
        %1 は先にここで定義されています
%1 redeclared as different kind of symbol
        %1 は違う種類のシンボルとして宣言しなおされました
%1 redefined
        %1 が再定義されました
%1 undeclared (first use this function)
         %1 が宣言されていません(この関数内部で初めて使われました)
%1 undeclared here (not in a function)
         %1 がここでは宣言されていません(関数の内部にありません)
%1 undeclared, outside of functions
         %1 が宣言されていません(関数外部で使用されています)
%1 used prior to declaration
        %1 は宣言する前に使われています
%1 was declared %2 and later %3
        %1 は %2 と宣言され、後で %3 と宣言されました
%1 was declared implicitly %2 and later %3
        %1 は以前に %2 と宣言され、後に %3 と宣言されました
%1 was not declared in this scope
         このスコープ(変数の有効範囲)で %1 は宣言されませんでした
%1 was previously implicitly declared to return %2
        %1 は以前の暗黙の宣言で %2 を返すように宣言されています
%1 with a value, in function returning void
        値を返さない関数に返し値を持った %1 があります
%1 with no argument
        %1 に引数がありません
(Each undeclared identifier is reported only once
         (宣言されていない変数や関数については１度だけ報告されます .
An argument type that has a default promotion
        An argument type that has a default promotion
At top level
         トップレベルで:
Compilation of header file requested
        ヘッダファイルをコンパイルせよといわれました
In function %1
         関数 %1 において:
Is a directory
        これはディレクトリです
No input files
        入力用ファイルがありません
No input files specified.
        入力用ファイルが指示されていません
```

No space left on device
　　ディスクがいっぱいです
No such file or directory
　　そのようなファイルやディレクトリは見つかりません
Permission denied
　　パーミッションがありません
Undefined symbol
　　以下のシンボルが未定義です
aggregate has a partly bracketed initializer
　　初期化の集合の要素がひとまとめにされています
argument %1 doesn't match prototype
　　引数 %1 はプロトタイプと一致していません
argument %1 may be clobbed by %2
　　引数 %1 は %2 によって壊されるかもしれません
argument to %1 is missing
　　%1 に対する引数が書かれていません
arguments given to macro %1
　　マクロの %1 に実引数が与えられました
arithmetic on pointer to an incomplete type
　　不完全な型のポインタで演算を行っています
array %1 initialized from non-constant array expression
　　配列 %1 は不定の配列によって初期化されています
array initialized from non-constant array expression
　　配列が不定の配列によって初期化されています
array size missing in %1
　　配列 %1 の大きさが指定されていません
array subscript is not an integer
　　配列の添字が整数でありません
assignment from incompatible pointer type
　　型が一致していないポインタからの代入が行われています
assignment makes integer from pointer without a cast
　　キャストせずにポインタを整数型の変数に代入しようとしています
assignment makes pointer from integer without a cast
　　キャストせずに整数型の変数をポインタに代入しようとしています
at this point in file
　　at this point in file
bad flag
　　bad flag
bad magic number
　　オブジェクトファイルではありません
bad parameter list specification for function %1
　　関数 %1 に誤りのある引数のリストが書かれています
badly punctuated parameter list in %1
　　%1 の引数リストの記述が不正です
braces around scalar initializer
　　スカラー変数を初期化する要素が{ }で囲まれています
break statement not within loop or switch
　　break がループ内やスイッチ文内で使われていません
built-in function %1 declared as non-function
　　標準関数の %1 が非関数として宣言されました
built-in function %1 used without declaration
　　標準関数 %1 が宣言なしで使われました
called object is not a function
　　関数でないものが関数として呼び出されています
can't match an empty parameter name list declaration.
　　can't match an empty parameter name list declaration.
can't open shared library %1
　　共有ライブラリ %1 がみつかりません
case label does not reduce to an integer constant
　　case ラベルの後に整数の定数式が書かれていません
case label not within a switch statement
　　case ラベルがスイッチ文内で使われていません
cast from pointer to integer of different size
　　ポインタをサイズの違う整数型に変換しています
character constant too long
　　文字定数が長すぎます
comparison between pointer and integer
　　ポインタと整数が比較されています

comparison is always 0 due to limited range of data type
　　データ型の表現範囲が限られているので比較結果が常に 0 になります
comparison is always 1 due to limited range of data type
　　データ型の表現範囲が限られているので比較結果が常に 1 になります
conflicting declarations of %1
　　%1 が二重に宣言されています
conflicting types for %1
　　%1 が二重に宣言されています
conflicting types for built-in function %1
　　標準関数の %1 と型が一致していません
constant value must be between -4096 and 4095
　　constant value must be between -4096 and 4095
continue statement not within a loop
　　continue がループ内で使われていません
control reaches end of non-void function
　　control reaches end of non-void function
data definition has no type or storage class
　　データ型の宣言に型名が書かれていません
decimal constant is so large that it is unsigned
　　10 進数定数が大き過ぎるので符合なし(unsigned)となります
declaration for parameter %1 but no such parameter
　　%1 という引数の型が宣言されましたが、そのような引数はありません
declaration of %1 shadows a parameter
　　関数の引数 %1 の宣言が関数内部で行われています
default label not within a switch statement
　　default がスイッチ文外で使われています
dereferencing pointer to incomplete type
　　不完全な型のポインタで参照しています
doesn't match non-prototype definition here
　　ここでの関数の定義と一致していません
duplicate case value
　　同じ値をもった case ラベルが使われています
elements of array %1 have incomplete type
　　配列 % の要素の大きさが決められません
empty #if expression
　　条件式が書かれていません
empty character constant
　　空の文字定数('')があります
excess elements in aggregate initializer after %1
　　%1 の後の初期化の集合の要素が多過ぎます
excess elements in array initializer after %1
　　%1 の後の配列を初期化する要素が多過ぎます
excess elements in scalar initializer after %1
　　%1 の後のスカラー変数を初期化する要素が多過ぎます
extern declaration of %1 doesn't match global one
　　%1 の extern 宣言はグローバルのものと合っていません
field %1 has incomplete type
　　フィールド %1 は不完全な型です
floating constant exponent has no digits
　　浮動小数点定数の指数部が数字でありません
for each function it appears in.)
　　各関数内で見つけられるごとに報告されません)
garbage at end of number
　　数字表現の後ろにゴミがあります
gcc version %1
　　gcc のバージョンは %1 です
implicit declaration of function %1
　　関数 %1 の暗黙の宣言はここで行われました
incompatible type for %1 of indirect function call
　　間接に呼び出した関数の第 %1 引数の型が矛盾しています
incompatible type for %2 of %1
　　関数 %1 の第 %2 引数は型が矛盾しています
incompatible types in assignment
　　代入する変数の型が一致していません
incompatible types in initialization
　　初期化において型の矛盾があります
incompatible types in initialization of %1
　　%1 の初期化に矛盾があります

incompatible types in return
関数の型と返り値の型が一致していません
initialization from incompatible pointer type
矛盾したポインタ型で初期化が行われています
initialization makes pointer from integer without a cast
初期化においてキャストせずに整数型をポインタに変換しています
initialization of %1 from incompatible pointer type
矛盾したポインタ型で %1 の初期化が行われています
initialization of %1 makes integer from pointer without a cast
%1 の初期化はキャストせずにポインタを整数型の変数に代入しています
initialization of %1 makes pointer from integer without a cast
%1 の初期化はキャストせずに整数型の変数をポインタに代入しています
initializer element for %1 is not constant
%1 の初期化の要素が定数ではありません
initializer element is not constant
初期値の要素が定数ではありません
initializer for scalar %1 requires one element
スカラー変数 %1 の初期化は１つの要素で行わなければなりません
integer constant is so large that it is unsigned
整数定数が大き過ぎるので符合なし(unsigned)となります
integer constant out of range
整数定数が範囲を越えています
integer overflow in expression
整数表現にオーバーフローがあります
invalid initializer
初期値に誤りがあります
invalid initializer for %1
%1 の初期化に誤りがあります
invalid lvalue in assignment
代入に無効な左辺値です
invalid lvalue in decrement
デクリメント演算子に無効な対象が書かれています
invalid lvalue in increment
インクリメント演算子に無効な対象が書かれています
invalid lvalue in unary %1
単項演算子 %1 の左辺値に誤りがあります
invalid macro name
マクロが無効です
invalid macro name %1
%1 というマクロは無効です
invalid operands to binary %1
二項演算子 '%1' に対するオペランドが無効です
invalid storage class for function %1
関数 %1 の記憶クラスは無効になります
invalid type argument of %1
単項演算子 `%1' の引数に誤りがあります
invalid use of array with unspecified bounds
大きさを指定していない配列が使われていますが無効です
invalid use of void expression
'void' が無効になっています
its scope is only this definition or declaration,
そのスコープ(変数の有効範囲)は、この宣言部分だけです
label %1 used but not defined
ラベル %1 が使われましたが定義されていません
large integer implicitly truncated to unsigned type
large integer implicitly truncated to unsigned type
linker input file unused since linking not done
リンクされなかったのでリンカが読み込んだファイルは使われませんでした
long, short, signed or unsigned invalid for %1
long, short, signed または unsigned は %1 には使えません
macro %1 used with just one arg
マクロの %1 を１個の引数で使用しています
macro %1 used with too many %2 args
macro %1 used with too many %2 args
macro %1 used without args
マクロの %1 が引数なしで使用されました
malformed floating constant
浮動小数点数の表現に誤りがあります

multi-character character constant
　　文字が複数の文字定数があります
multiple parameters named %1
　　%1 という引数の名前が複数書かれています
multiple storage classes in declaration of %1
　　%1 の宣言に複数の記憶クラスが書かれています
multiply defined
　　多重に定義されています
no semicolon at end of struct or union
　　構造体や共用体の宣言に誤りがあります
nondigits in number and not hexadecimal
　　数字の中に 0 〜 9 以外の文字が含まれ、それは 16 進数表現でもありません
number of arguments doesn't match prototype
　　引数の個数が宣言されたものと合っていません
numeric constant contains digits beyond the radix
　　基数以上の数を含んだ定数が書かれています
ordered comparison of pointer with integer zero
　　ポインタと整数定数の 0 との比較が命令されました
overflow in constant expression
　　定数表現にオーバーフローが起こっています
overflow in implicit constant conversion
　　暗黙の定数型変換でオーバーフローが起こりました
parameter %1 has incomplete type
　　引数 %1 は型が不完全です
parameter %1 has just a forward declaration
　　引数 %1 は前方参照宣言の形になっています(;がついています)
parameter %1 is initialized
　　引数 %1 が初期化されています
parameter has incomplete type
　　引数が不完全です
parameter name omitted
　　引数の名前が書き忘れられています
parameter names (without types) in function declaration
　　関数の宣言部分に型名のない引数があります
parm types given both in parmlist and separately
　　引数の型が( )内と( )と{ の間の両方で宣言されています
parse error at %1
　　%1 に構文上の間違いがあります
parse error at end of input
　　終りの閉じかっこがありません(} が足りません)
parse error before %1
　　%1 の前に構文上の間違いがあります
parse error before character %1
　　文字 %1 の前に間違いがあります
parse error before character constant
　　文字定数の前に構文上の間違いがあります
parse error before string constant
　　文字列定数の前に構文上の間違いがあります
passing %2 of %1 from incompatible pointer type
　　関数 %1 の第 %2 引数は仮引数とポインタの種類が異なるものを渡しています
passing %2 of %1 makes integer from pointer without a cast
　　関数 %1 の第 %2 引数渡しはキャストせずにポインタを整数型に変換しています
passing %2 of %1 makes pointer from integer without a cast
　　関数 %1 の第 %2 引数渡しはキャストせずに整数型をポインタに変換しています
pathname missing
　　パス名がありません
pointer value used where a floating point value was expected
　　浮動小数点定数が期待されるところにポインタ値が使われています
possible real start of unterminated constant
　　閉じられていない文字列の始まりはこの行と思われます
premature EOF
　　premature EOF
previous declaration of %1
　　%1 はこの行で一度宣言されています
previous external decl of %1
　　以前の %1 の外部宣言はここで行われました
previous implicit declaration of %1
　　%1 の暗黙の宣言はここで行われました

prior parameter's size depends on %1
　　先の引数の大きさは %1 に依存しています
prototype declaration
　　プロトタイプの宣言はここで行われました
prototype for %1 follows and %2
　　関数 %1 の %2 番目の引数の型が
redeclaration of %1
　　%1 が再宣言されました
redefinition of %1
　　%1 の再定義が行われています
request for member %1 in something not a structure or union
　　構造体や共用体でないものでメンバ %1 の参照が行われています
return from incompatible pointer type
　　返り値のポインタの型が矛盾しています
return makes integer from pointer without a cast
　　return の返り値はキャストせずにポインタを整数型に変換しています
return makes pointer from integer without a cast
　　return の返り値はキャストせずに整数型をポインタに変換しています
return type for %1 changed to integer type
　　%1 の返り値の型は整数型に変えられました
return-type defaults to %1
　　返り値はデフォルトで %1 になります
size of array %1 has non-integer type
　　配列 %1 の大きさが整数で指定されていません
size of array %1 is negative
　　配列 %1 の大きさが負数になっています
size of variable %1 is too large
　　変数 %1 はサイズが大き過ぎます
static declaration for %1 follows non-static
　　%1 の static 宣言の後に static クラスにない型が書かれています
storage class specified for parameter %1
　　引数 %1 は記憶クラス指定子(static,extern など)が書かれています
storage size of %1 isn't constant
　　%1 を確保する記憶領域の大きさが一定ではありません
storage size of %1 isn't known
　　%1 を確保する記憶領域の大きさがわかりません
stray '\' in program
　　エスケープ文字 '\' が ' や " で囲まれていないところで使用されました
structure defined inside parms
　　引数の宣言の中で構造体が定義されています
structure has no member named %1
　　構造体に %1 という名前のメンバがありません
subscripted value is neither array nor pointer
　　添字がつけられている変数は配列でもポインタでもありません
suggest parentheses around assignment used as truth value
　　代入した結果を真偽値として使っています
switch quantity not an integer
　　switch 文の式は整数値を結果とするものでなければなりません
syntax error before %1
　　%1 の前は規則に合っていません
this is the first entry for that value
　　ここがその値をもつ最初の case ラベルです
this is the location of the previous definition
　　ここが先に定義が行われた箇所です
too few arguments to function %1
　　関数 %1 の引数の数が足りません
too many arguments for function %1
　　関数 %1 の実引数が多過ぎます
too many arguments to function %1
　　関数 %1 の引数が多すぎます
two or more data types in declaration of %1
　　２回かそれ以上 %1 のデータ型の宣言がされています
type mismatch in implicit declaration for built-in function %1
　　標準関数 %1 の暗黙の宣言と型が合っていません
type mismatch with previous external decl
　　以前の外部宣言と型が合っていません
type mismatch with previous implicit declaration
　　以前の暗黙の宣言と型が合っていません

type of %1 defaults to %2
　　%1 の型はデフォルトで %2 になります
type of formal %1 is incomplete
　　第 %1 番目の仮引数の型が不完全です
undefined or invalid # directive
　　定義されていないか無効なプリプロセッサ疑似命令が書かれています
union has no member named %1
　　共用体に %1 という名前のメンバがありません
unknown escape sequence %1
　　%1 というエスケープシーケンスがわかりません
unreachable code at beginning of switch statement
　　スイッチ文の始めから届かない範囲に命令があります
unrecognized option %1
　　%1 というオプションが認識できません
unsigned value >= 0 is always 1
　　unsigned のついた値 >= 0 は常に真です
unterminated %1 conditional
　　条件節 %1 が閉じられていません
unterminated character constant
　　文字定数が閉じられていません
unterminated comment
　　コメントの終りの印がありません
unterminated macro call
　　マクロの呼び出しが閉じられていません
unterminated string or character constant
　　文字列が閉じられていません
unused variable %1　　変数 %1 は使われていません
use of %1 length character with %2 type character
　　修飾文字 %1 が変換文字 %2 といっしょに使われています
value computed is not used
　　計算された値が使われていません
variable %1 has initializer but incomplete type
　　変数 %1 は初期化の要素は不完全な型です
variable %1 may be clobbed by %2
　　変数 %1 は %2 によって壊されるかもしれません
variable or field %1 declared void
　　変数かビットフィールドの %1 が void と宣言されました
variable-size type declared outside of any function
　　大きさが変わる型が関数の外部で宣言されています
variable-sized object may not be initialized
　　大きさが変わる対象を初期化してはいけません
virtual memory exhausted
　　仮想記憶に余裕がありません
void value not ignored as it ought to be
　　無視されなければならない戻り値が無視されていません
which is probably not what you want.
　　たぶんそれはあなたが望んでいることではないでしょう
wrong type argument to increment
　　変数の型がインクリメントできない型になっています
wrong type argument to unary %1
　　単項演算子 %1 の引数に誤りがあります