

プログラミングを始めよう

プログラミングツールの使い方

大浦 真@ LILO

`ohura@netfort.gr.jp`

目次

- なぜ、プログラミング?
- どうやって、プログラミング?
- エディタの用意
- まずは、gcc
- make 登場
- RCS で リビジョン管理
- gdb で バグ取り

目次

- なぜ、プログラミング?
- どうやって、プログラミング?
- エディタの用意
- まずは、gcc
- make 登場
- RCS で リビジョン管理
- gdb で バグ取り
- 拙作も紹介したりして

なぜ、プログラミング?

- PC-UNIX 文化を支える豊富なフリーソフト
ex. Apache, sendmail, bind, Mozilla, XFree86, Gnome, KDE, gimp,...
- その影には、必ず開発環境がある。
ex. gcc, make, CVS, RCS, gdb,...
- これらの開発環境も PC-UNIX を支える優秀なフリーソフト
- 普通のユーザも簡単に開発をする側になれる。

どうやって、プログラミング?

- 普通のディストリビューションなら、インストール後すぐに開発環境を使うことができる。
- どの言語を試してみるか、考える。
C 言語 定番中の定番。
Perl スクリプト言語。テキスト処理とか、CGI とか。
Ruby, Java,... その他いろいろ。
- 後は、いくつかのプログラミングツールの使い方を覚える。
- エディタ, gcc, make, RCS, gdb

エディタの用意(1)

プログラミングをするためのエディタを用意しよう。

- どんなエディタを使うか。

エディタの用意(1)

プログラミングをするためのエディタを用意しよう。

- どんなエディタを使うか。
- なんでもいい。使いやすいものを。
ex. vi, GNU Emacs,...
- エディタによっては、プログラミング用の機能が付いていることがある。

エディタの用意(2)

例えば、XEmacs。

```
xemacs: /usr/bin/xemacs [21.4 (patch 1) "Coryleft" XEmacs Lucid] b2cmain.c
Functions ファイル 編集 View Cncls ツール オプション パッチャ ヘルプ
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News
gencode.c b2cmain.c
#include <unistd.h>
#include <getopt.h>
#include "b2c.h"

int gyparse();

int nodec = 0;
FILE *yyin;
FILE *templ;

int main(int argc, char *argv[])
{
    char c;

    struct option long_option[]={
        {"help", 0, 0, 'h'},
        {"version", 0, 0, 'v'},
        {0, 0, 0, 0}
    };

    while((c = getopt_long(argc, argv, "hv", long_option, 0)) != -1){
        switch(c){
            case 'h':
                printhehelp();
                exit(EXIT_SUCCESS);
            case 'v':
                printversion();
                exit(EXIT_SUCCESS);
            default:
                break;
        }
    }
}
```

Ja/EUC-----XEmacs: b2cmain.c 3:28. (C Font)-----L30--41%

まずは、gcc(1)

- gcc:GNU compiler collection
- PC-UNIX 上で動くほぼ唯一の C compiler
- 実際には、C だけではなく、C++, Objective-C, Fortran-77, Java, Chill, Pascal のコンパイラが含まれている。
- 現在最新バージョンは、2.95.3。近い将来、3.0 が出るらしい。
- フリーソフトの大半は、C 言語で書かれている。プログラミングをしなくても、C 言語でプログラミングをしなくても、使い方くらいは覚えておいて損はない。
- <http://www.gnu.org/software/gcc/gcc.html>

まずは、gcc(2)

サンプルの C のソース — hello.c

```
#include <stdio.h>

int main()
{
    printf("Hello, World!!\n");

    exit(0);
}
```

定番中の定番。Hello, World!! を表示する C 言語のプログラム。
これを、エディタで入力して hello.c という名前で保存。

まずは、gcc(3)

そして、コンパイル。

```
% gcc -o hello hello.c
```

実行!!

```
% ./hello  
Hello, World!!  
%
```

となったら、無事コンパイルできた。
('./' は忘れずに。)

まずは、gcc(4)

コマンドラインオプションの説明。(gcc には数限りないオプションがあるので、重要なものののみ。)

- o 出力ファイル名の指定。これを指定しないと出力ファイル名は、'a.out' になる。
- Wall 全ての警告を出力する。
- O2 コンパイル時に最適化を行う。
- c コンパイルのみを行う。(make の時に使う。)
- g デバッグ情報を組み込む。

make 登場(1)

ファイルが複数になったらどうしよう。例えば、
— hello-1.c

```
void printhello();

int main()
{
    printhello();

    exit(0);
}
```

make 登場 (2)

— hello-2.c

```
#include <stdio.h>

void printhello()
{
    printf("Hello, World!!\n");
}
```

make 登場 (3)

このままでも、一応、

```
% gcc -o hello hello-1.c hello-2.c
```

とするか、

```
% gcc -c hello-1.c  
% gcc -c hello-2.c  
% gcc -o hello hello-1.o hello-2.o
```

とすることによってコンパイルはできる。

make 登場 (4)

しかし、

- 一つのファイルを書き換えるだけで全てのファイルをコンパイルする必要が出てくる。
- 分割してコンパイルするとしても、どのファイルが更新されたか把握していないと結局同じ。

そこで make 登場。コンパイルする規則を記述する Makefile を用意する。

make 登場 (5)

こんな Makefile を書いてみる。
— Makefile

```
CC=gcc
CFLAGS=-Wall -O2 -g

hello-1 : hello-1.o hello-2.o
hello-1.o : hello-1.c
hello-2.o : hello-2.c

clean :
    rm -f hello-1 *.o
```

(注意!! 9行目の 'rm' の前は TAB でなければならない。)

make 登場 (6)

そして、make 実行。

```
% make
gcc -Wall -O2 -g -c -o hello-1.o hello-1.c
gcc -Wall -O2 -g -c -o hello-2.o hello-2.c
gcc hello-1.o hello-2.o -o hello-1
% ls hello-1
hello-1
%
```

コマンド一つで hello-1 ができた!!

make 登場 (7)

clean というターゲットを作ったので、いらぬファイルを消すこともできる。

```
% ls
Makefile  hello-1.c  hello-2.c  hello.c
hello-1   hello-1.o  hello-2.o
% make clean
rm -f hello-1 *.o
% ls
Makefile  hello-1.c  hello-2.c  hello.c
%
```

make 登場 (8)

- ‘:’ の前はターゲット名。‘:’ の後ろはそのターゲットに依存するターゲットの名前。
- 次の行に、そのターゲットを作るためのコマンドを記述。
- ターゲットより依存ファイルの方が新しくなった時のみコマンドは実行される。
- C 言語ファイルからオブジェクトファイルを作る、あるいは、オブジェクトファイルから実行形式を作るためのコマンドは、make の方で知っていてくれる。その時のコマンド名は ‘CC’、コマンドラインオプションは ‘CFLAGS’ で指定する。
- make コマンドの引数には、ターゲット名を指定する。省略されたら、Makefile の一番上のターゲットを作成する。

make 登場 (9)

- プログラミングでなくても make は使える。例えば、 $\text{T}_{\text{E}}\text{X}$ 。
- 今、プレゼンテーションしているこのファイルの Makefile もある。

RCS でリビジョン管理

- 少し長いプログラムを書いていくと、修正した部分が気に入らなくて元に戻したくなることが良くある。
- そのためにいちいちバックアップをとっておくのは面倒。
- 版の管理をしたくなる。
- そのために、RCS(revision control system)
- RCS は rcs, ci, co, rlog,... などのコマンドの集まり。
- 例えば、hello.c に対して、RCS/hello.c,v というファイルで版の管理を行う。
- このファイルには、今までの全ての版の情報や、版を登録した時に入力した log も全て記録されている。

gdb でバグ取り (1)

- プログラムはいつも思い通りに動くとは限らない。というか、動かない方が圧倒的に多い。
- 原始的なやり方としては、怪しい部分に printf 文などを入れて、変数の値を表示させる方法がある。
- それとは別に、デバッガを使って、プログラムを one step ずつ実行させてリアルタイムに変数の値の変化を見ていく方法もある。
- それに使えるのが、gdb(GNU debugger)
- gdb を使うためには、gcc に -g オプションが必要。
- ソースを見ながら使える gdb のフロントエンドを使うと便利。
ex. Emacs の gdb モード, DDD(Data Display Debugger),...

gdb でバグ取り (2)

XEmacs 上でも、gdb が動く。

```
XEmacs: /usr/bin/xemacs [21.4 (patch 1) "Copyleft" XEmacs Lucid] *gdb-b2c*
ファイル 編集 View Cmds ツール オプション パッチ Complete In/Out Signals ヘルプ
*gdb-b2c*
3: l = 1
2: j = 0
1: i = 0
(gdb) n
3: l = 1
2: j = 0
1: i = 0
(gdb) |

[--]Ja/EUC--*-XEmacs: *gdb-b2c* 5:38. (Inferior GDB Frame:run)----L188
    vartbl[i].name);
} else {
    fprintf(stdout, "double %s[%d];\n",
            vartbl[i].name, vartbl[i].max);
}
break;
case T_STR: /* STRLEN is defined in libb2c.h */
=>if (vartbl[i].max == 1) {
    fprintf(stdout, "char %s[STRLEN];\n",
            vartbl[i].name);
} else {
    fprintf(stdout, "char %s[%d][STRLEN];\n",
            vartbl[i].name, vartbl[i].max);
}
break;

[--]Ja/EUC----XEmacs: gencode.c 'codeout' 5:38. (C Font)----L122--30%
```


b2c の紹介 (1)

- b2c : 大浦 真氏が開発中の BASIC から C 言語へのトランスレータ。
- 「プログラミングを始めたい。昔、N88 BASIC は少しやったことがある。」という人にはお勧めかな。

b2c の紹介 (2)

例えば、

```
input a
do
    if a = int(a/2)*2 then
        b = a/2
    else
        b = a*3+1
    end if
    print b
    a=b
loop until a=1
end
```

という BASIC のソースが、

b2c の紹介 (3)

```
#include <stdio.h>
#include <string.h>
#include <libb2c.h>
int main()
{
    float a;
    float b;
    scanf("%f", &a);
    do {
        if (a == integer(a / 2) * 2) {
            b = a / 2;
        } else {
            b = a * 3 + 1;
        }
        printf("%.7g\n", b);
        a = b;
    }
    while (!(a == 1));
    exit(0);
}
```

というような C 言語のソースに変換される。

b2c の紹介 (4)

- ただいま、ユーザ募集中。
- ただし、現在まだ開発初期段階。
- Web ページ
`http://www.netfort.gr.jp/~ohura/develop.shtml`
- deb パッケージや rpm パッケージもある。

終わりに

- さあ、プログラミングを始めてみよう。
- いいものができたら、全世界に向けて公開しよう。

終わりに

- さあ、プログラミングを始めてみよう。
- いいものができたら、全世界に向けて公開しよう。
- deb パッケージ作成は請け負います。

終わりに

- さあ、プログラミングを始めてみよう。
- いいものができたら、全世界に向けて公開しよう。
- deb パッケージ作成は請け負います。

ありがとうございました。