Makefileことはじめ

Makefile について理解が足りてないのでメモ Ruby の C 拡張をいじるに当たって、避けて通れないので、ここいらで把握する

make って何?

```
make(メイク) は、プログラムのビルド作業を自動化するツール。
コンパイル、リンク、インストール等のルールを記述したテキストファイル (Makefile) に従って、これらの作業を自動的に行う。
複雑に関連し合ったファイルの依存関係を解決するのが make の長所である。
```

make - Wikipedia (http://ja.wikipedia.org/wiki/Make)

サンプルコード

```
#include <stdio.h>

void test(){
  printf("test");
}

int main(){
  test();
}
```

ビルド、動作確認

```
$ gcc test.cp
$ ./a.out
test
```

Makefile を作ってみる

```
test: test.o
```

```
$ make
cc     -c -o test.o test.c
cc   test.o    -o test
$ ./test
test
```

- cc は、gcc のエイリアス
- -c オプションは、リンカを起動しないオプション

既にある実行ファイルや、オブジェクトファイルは生成しない もっかい make を実行すると、

```
make: `test' is up to date.
```

と、言われて何もしない

リンカ?

```
リンケージエディタ（リンカ(linker)、連係編集プログラムとも）とは
機械語のプログラムの断片を結合し実行可能なプログラムを作成するプログラムのことである。
例として、c言語では、ソースファイルをコンパイルするとオブジェクトファイルが生成される。
それに他のオブジェクトファイルやライブラリを結合して1つのプログラムが完成する。
この結合（リンクという）の際リンケージエディタが使われる。
```

リンケージエディタ - Wikipedia
(http://ja.wikipedia.org/wiki/%E3%83%AA%E3%83%B3%E3%82%B1%E3%83%BC%E3%82%B8%E3%82%A8%E3%83%87%E3%82%A3%E3%82%BF)

上記だと、

- まずは test.c のオブジェクトファイル test.o を生成
- リンカを使って、test.o から、実行可能ファイル test を生成している

Makefileの解説

基本的な Makefileのフォーマット

```
ターゲット(作りたいファイル名): 依存ファイル, 依存ファイル...
```

```
  コマンド
  コマンド
  ...
```

```
test: test.o
```

ルール

上記の場合、test というターゲット(実行可能ファイル) を作るのに、test.o が依存しているよ! と書いている このひとかたまりを **ルール** と呼ぶ

ターゲット

```
$ make ${ターゲット名}
```

Makefile のターゲットのルールを実行する デフォルトは、Makefileの一番上のターゲットのルールを実行する なので、今回の場合、make コマンドを実行すると、test ターゲットのルールが実行される

コマンド

ターゲット実行時に、実行されるコマンドを書く 省略した場合、暗黙のルール(後述)に従ったコマンドが実行される コマンドと行頭の間の空白は **かならず tab であること**

make の処理の流れ

make は上記の Makefile を読んで、以下の様な処理を行う

1. 実行ファイル test を生成するのに、test.o が必要と理解
2. test.o は、test.c からコンパイルされるはずと考えて、test.c をリンカなしでコンパイル
3. test.o を使って、実行ファイル test を生成

暗黙のルール

上ように、make は、コンパイラとかソースコードの場所を示したルールやコマンドを書かなくても、大体適当にやってくれる この慣習的に、適当にやってくれるルールを **「暗黙のルール」** と言う

- GNU make 日本語訳(Coop編) - 暗黙ルールの利用 (http://www.ecoop.net/coop/translated/GNUMake3.77/make_10.jp.html)

make clean の実装

Makefileには、慣習としていくつかのターゲットが存在すべきらしい make clean とか、make install とか

- GNU make 日本語訳(Coop編) - 慣習的なMekifile#ユーザーのための標準ターゲット (http://www.ecoop.net/coop/translated/GNUMake3.77/make_14.jp.html#SEC117)

いろいろあるけど、試しに clean を実装してみる

```
`clean'
      全ファイルを(普通はプログラムのビルド時に作った)カレントディレクトリから削除します。
```

```
test: test.o
clean:
    rm test test.o
```

単純に、make で生成される実行ファイルと、オブジェクトコードを削除する

```
$ make clean
rm test test.o
```

エラー無視オプション

例えばこんな Makefile 削除をしてから、何かしらの処理を行いたい場合

```
test: test.o
clean:
    rm test test.o
    echo "test" # 削除の後のなんらかの処理
```

だけど、例えば test.o がない場合、echo "test" は実行されない

```
$ make clean
rm test test.o
rm: test.o: No such file or directory
make: *** [clean] Error 1
```

---

make clean の場合、とにかくファイルを消すことが目的なので、全てのファイルが存在しなくても、消せるやつだけ消せれば良い そういう時に、No such file or directory でエラーが発生して、処理が途中で止まっても困る

コマンドの前に "-" を入れることで、その行でエラーが発生しても、正常終了したと見なしてくれる

```
test: test.o
clean:
    -rm test test.o
    echo "test" # 削除の後のなんらかの処理
```

こうすると、echo "test" が実行される

```
make clean
rm test test.o
rm: test.o: No such file or directory
make: [clean] Error 1 (ignored)
echo "test"
test
```

擬似ターゲット

ターゲットは、基本的にターゲット名と同名のファイルを生成する処理 なので、make は、ターゲットと同名のファイルが既にあると(依存関係がある時以外は)処理を行わない

なので、 **clean というファイルがあると、make clean は実行されない!**

```
$ touch clean
$ make clean
make: `clean' is up to date.
```

.PHONY ${ターゲット名} という記載で、そのターゲットは、ファイルは生成しないという事を make に知らせることができる

```
test: test.o
.PHONY: clean
clean:
    -rm test test.o
    echo "test" # 削除の後のなんらかの処理
```

依存関係のあるコードを make

```
#ifndef __TEST_H__
#define __TEST_H__
void test();
#endif
```

```
#include <stdio.h>
void test(){
  printf("test");
}
```

```
#include <test.h>
int main(){
  test();
}
```

まずは、手動でコンパイル

test.c をコンパイルしてから、main.c をコンパイルする

```
$ gcc -c test.c -o test.o -I./
$ cc -c main.c
$ cc main.o test.o -o test
```

Makefile を書いてみる

```
test: test.o main.o # コードオブジェクトを順序依存に書く
.PHONY: clean
clean:
    -rm test test.o main.o
    echo "test" # 削除の後のなんらかの処理
```

make

```
$ make
cc    -c -o test.o test.c
test.c:1:18: error: test.h: No such file or directory
make: *** [test.o] Error 1
```

test.h が見つからないと怒られる

マクロ

カレントディレクトリを、インクルードパスとして -I オプションで追加する必要がある

```
CFLAGS=-I./
test: test.o main.o # コードオブジェクトを順序依存に書く
.PHONY: clean
clean:
    -rm test test.o main.o
    echo "test" # 削除の後のなんらかの処理
```

無事成功

```
$ make
cc -I./   -c -o test.o test.c
cc -I./   -c -o main.o main.c
cc   test.o main.o   -o test
```

CFLAGSは、暗黙ルールの中で定義されている、Cコンパイル時に追加するオプションを格納する変数 このような変数を **マクロ** と呼ぶ

- 他にも色んなマクロがある → GNU make 日本語訳(Coop編) - 暗黙ルールの利用 (http://www.ecoop.net/coop/translated/GNUMake3.77/make_10.jp.html#SEC92)

参考

- Man page of MAKE (http://linuxjm.sourceforge.jp/html/GNU_make/man1/make.1.html)
- GNU make 日本語訳(Coop編) - makeコマンドの概要 (http://www.ecoop.net/coop/translated/GNUMake3.77/make_1.jp.html)
- 仕事で使える魔法のLAMP（8）：makeを使ってソフトウェアをビルドしてみよう - @IT (http://www.atmarkit.co.jp/ait/articles/1106/07/news131.html)
- MAKE のマクロ - RAD Studio (http://docwiki.embarcadero.com/RADStudio/XE4/ja/MAKE_%E3%81%AE%E3%83%9E%E3%82%AF%E3%83%AD)
- Make チュートリアル (http://homepage3.nifty.com/kaku-chan/make/index.html)
- 自動化のためのGNU Make入門講座 (http://objectclub.jp/community/memorial/homepage3.nifty.com/masarl/article/gnu-make.html)