makefile 課題

学籍番号 045713C：大城 和也

平成 17 年 5 月 10 日

1 課題内容

- 必須：全ての level にトライし、レポートとしてまとめよ。
- オプション：gcc 以外にどのような使い道があるか？検討/調査/実験し，その結果を報告せよ。

2 必須

2.1 level1

level1：サンプルプログラムをダウンロードし、make を実行せよ。make 実行時、端末上にその動作が表示された内容を記録し、その内容が何を意味するのか説明せよ。

2.1.1 表示内容

```
[nw0413: /X-file/Expander/make-sample] j04013% make
cc -c -o sample.o sample.c
cc -c -o add.o add.c
cc -c -o multi.o multi.c
cc sample.o add.o multi.o -o sample
```

2.1.2 説明

この表示の内容はまず、make というコマンドを用いる事で Makefile を呼び出す。Makefile の内容は

```
sample:  sample.o add.o multi.o
```

となっておりこれは sample というプログラムを作るのに sample.o、add.o、multi.o というオブジェクトが必要という事を示してる。しかし、このディレ

クトリ内にこれらのオブジェクトはない。よって make は*.c から*.o を作り出す事を推測して実行していると考えられる。

2.2 level2

level2：どれか 1 つ以上のソースファイルを編集・保存し、make を実行せよ (コンパイルが通るのであれば編集内容は問わない)。編集対象毎に make の動作は異なるはずである。どのファイルを編集するとどう動作するのかを確認せよ。

2.2.1 表示内容

sample.c を編集後、make を実行したとき

```
[nw0413: /X-file/Expander/make-sample] j04013% make
cc -c -o sample.o sample.
cc sample.o add.o multi.o -o sample
```

add.c と multi.c を編集後、make を実行したとき

```
[nw0413: /X-file/Expander/make-sample] j04013% make
cc -c -o add.o add.c
cc -c -o multi.o multi.c
cc sample.o add.o multi.o -o sample
```

2.2.2 説明

この 2 つの出力を見てわかるように内容が変更された C 言語ソースのみが初めにコンパイルされその後にターゲットとなるプログラムを生成している。これは先の level1 でコンパイルされこの時点で何も変更されていなかった部分のコンパイルを省略しているということである。このように makefile は変更された部分だけをコンパイルすることで効率を図ることができ複数のファイルが必要な大きなプログラムを作る際に非常に便利である。

2.3 level3

level3：touch コマンドにより 1 つ以上のソースファイルのタイムスタンプを更新し、make を実行せよ。この時の make の動作は level2 と比較して同じはずである。何故このような動作になっているのか考察せよ。

2.3.1 表示内容

multi.c を touch コマンドを用いて更新した後 make を実行したとき

```
[nw0413: /X-file/Expander/make-sample] j04013% make
cc -c -o multi.o multi.c
cc sample.o add.o multi.o -o sample
```

2.3.2 説明

この時の make の動作は初めにコンパイルされているソースこそ違えど level2 の時と同じとなっている。複数のファイルに touch コマンドを適用した時も level2 の複数ファイルを編集した時と同じ動作になる。

level2 において内容を書き換えたときそのタイムスタンプは更新されている。これはつまり make がソースとオブジェクトとのタイムスタンプを比較して違っているならこのソースの内容は更新されたとみなされてコンパイルを行っていると考えられる。

2.4 level4

level4：サンプルには幾つかの Makefile 例があり、makefile2 は詳細動作まで記述した例である。makefile2 の source や command を削除し、make を実行せよ。この例では全く同一の動作になっているはずである。これは make に推論機能が実装されているためである。各自削除した source/command について、具体的に何が推論によりとらわれているのかを示せ。

2.4.1 makefile2

編集前

```
01:[nw0413: /X-file/Expander/make-sample] j04013% cat makefile2
02:# 詳            動            作            細            例
03:CC = gcc
04:CFLAGS = -Wall -O2
05:
06:sample:  sample.o add.o multi.o
07:  $(CC) sample.o add.o multi.o -o sample
08:
09:sample.o:  sample.c
10:  $(CC) -c sample.c
11:add.o:  add.c
12:  $(CC) -c add.c
13:multi.o:  multi.c
14:  $(CC) -c multi.c
15:
16:clean:
17:  rm -f *.o sample
```

編集前 makefile2 を実行したとき

```
01:[nw0413: /X-file/Expander/make-sample] j04013% make -f makefile2
02:gcc -c sample.c
03:gcc -c add.c
04:gcc -c multi.c
05:gcc sample.o add.o multi.o -o sample
```

2.4.2 説明

7 行目のコマンドを削除した場合、make を実行しても初めの時と変わらなかった。これは make がこのターゲットと指定されたソースからそのコンパイルの方法を推論したと考えられる。

次に 10 行目のコマンドを削除してみた、このとき makefile2 を実行すると結果の 2 行目にあたる所が

```
gcc -Wall -O2 -c -o sample.o sample.c
```

変化していた。これも make の推論機能のおかげでコンパイルは通っているが先ほどと違いコンパイル方法が異なるのはなぜか。これは make の推論がデフォルトの表現を用いて表すことに起因している、初めのマクロ定義における CC とは C コンパイラのことでこのマクロを gcc に置き換えているため

初めは gcc となる次に CFLAGS というマクロだがこのマクロは CC の引数として用いられるためコマンドをちゃんと指定しなかったこのコンパイル方法に関しては-Wall -02 というオプションが付け加えられたと考えられる。この行以外にも 12 行、14 行を削除した場合も同様のことが言える。

次に 9 行目のソースの sample.c だけを削除して make を実行すると変化はなかった。同様に 11 行、13 行のソースを削除しても変化はなかった。これは make が.o のファイルと.c とのファイルの関係がわかっているためである。しかし、6 行目のソースを削除するとコンパイル時に

```
[nw0413: /X-file/Expander/make-sample] j04013% make -f makefile2
gcc sample.o add.o multi.o -o sample
gcc:  sample.o:  No such file or directory
gcc:  add.o:  No such file or directory
gcc:  multi.o:  No such file or directory
gcc:  no input files
make:  *** [sample] Error 1
```

といったエラーが発生した。これは sample というプログラムが sample.o、add.o、multi.o からなることを make では推論しかねるためである。これらからわかるように、もし make が推論できないようなときにはしっかりとそのコマンドやソースを書くことが必要となる。

2.5 level5

level5：参考資料での clean は大抵の makefile に定義されている内容である（コマンドやその引数は必要に応じてマクロを使うことが多い）。これは何をしているのか。Level1 でダウンロードした Makefile に類似の機能を記述し、実行せよ。その際、clean 実行前後で何がどう変わるのかを示せ。

2.5.1

Makefile に clean を追加

```
[nw0413: /X-file/Expander/make-sample] j04013% cat Makefile
sample:  sample.o add.o multi.o

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

2.5.2 説明

これを見てわかるように clean がターゲットとなっている所にはソースが指定されておらず、コマンドだけの表示となっている。よって clean は % make clean とターゲットを指定して行う。clean のコマンドは % rm -f *.o *〜 となっておりこれは拡張子が.o のファイルと〜が最後についているファイル、つまり make を実行した際にできた中間ファイルを削除するという意味である。よって clean 実行前には残っている add.o、multi.o、sample.o などが clean 実行後にはなくなっている。