Um ficheiro *makefile* contém uma sucessão de regras encadeadas.

Uma regra divide-se em três áreas: objetivo, dependências e comandos, dispostos da seguinte forma:

```
objetivo: dependências
<TAB>comando
```

Uma regra interpreta-se do seguinte modo: se alguma **dependência** for mais nova – tiver uma data de criação ou modificação mais recente – que o **objetivo**, o **comando** será executado para atualizar o objetivo.

As dependências são ficheiros originais – ficheiros com código fonte – ou, por sua vez, são objetivos de outras regras.

```
1  main.elf: main.o stack.o
2        gcc main.o stack.o -o main.elf
3
4  main.o: main.c
5        gcc -c main.c -o main.o
6
7  stack.o: stack.c stack.h
8        gcc -c stack.c -o stack.o
9
10 clean:
11       rm -rf *.o main.elf
```

A invocação simples do programa **`make`**, procura o ficheiro com nome **`makefile`** e processa a primeira regra. Neste caso a regras das linhas 1 e 2.

```
$ make
```

Começa por verificar as dependências. As que forem objetivos de outras regras, são processadas segundo essas regras – neste caso **`main.o`** e **`stack.o`** são objetivos de outras regras. Depois de processadas as dependências, são comparadas as datas das dependências com a data do objetivo, neste caso comparadas as datas de **`main.o`** e **`stack.o`** com a data de **`main.elf`**. Se alguma dependência for mais recente que o objetivo o comando **`gcc main.o stack.o -o main.elf`** é executado.

Se as dependências não forem objetivos de outras regras, são ficheiros originais. Neste caso passa-se logo para a comparação de datas. Na segunda regra – linhas 4 e 5 – é comparada a data de **`main.o`** – objetivo – com a de **`main.c`** – dependência – se esta for mais recente então é executado o comando **`gcc -c main.c -o main.o`**.

Gerar todos os objetivos presentes no **`makefile`** independentemente das datas:

```
$ make -B
```

Especificar a regra que se pretende aplicar. Neste caso apenas a regra **`clean`** que elimina os ficheiros intermédios.

```
$ make clean
```

Especificar um ficheiro com nome diferente de **`makefile`**, por exemplo, **`makelib.`**

```
$ make -f makelib
```

No *makefile* podem usar-se artifícios de escrita para agilizar a sua definição, designadamente: variáveis; variáveis automáticas; regras implícitas.
A variável **CFLAGS** definida na linha 3 é usada para especificar opções de compilação comuns. É utilizada nas linhas 8 e 11.
A variável **OBJECTS**, aplicada como dependência da regra das linhas 7 e 8 permite especificar os ficheiros que irão ser usado para gerar o executável **main**.
Para os objetivos terminados em **.o**, para os quais exista uma dependência com o mesmo nome e terminada em **.c**, e não exista um regra explicita, é aplicada a regras implícita das linhas 10 e 11.
Utilizando regras implícitas dispensa-se a escrita de uma regras explicita para cada ficheiro fonte, o que se traduz em grande simplificação nos projetos extensos.

```
1  CC = gcc
2
3  CFLAGS = -Wall
4
5  OBJECTS = main.o stack.o
6
7  main: $(OBJECTS)
8        $(CC) $(CFLAGS) $^ -o $@
9
10 %.o: %.c
11       $(CC) $(CFLAGS) -c -o $@ $<
12
13 clean:
14       rm -rf *.o main
```

Variáveis automáticas

$@	nome do objetivo
$^	todas as dependências
$<	nome da primeira dependência
$?	nome das dependência mais recentes que o objetivo
$*	nome do objetivo sem a extensão

Referências

http://www.gnu.org/software/make/manual/make.html