Makefile	Note
I makefile sono un modo semplice per organizzare la compilazione del codice. se ad esempio vuoi lanciare due file .c inclusi in una libreria .h dovresti ogni volta scrivere il comando di compilazione: **gcc -o hellomake hellomake.c hellofunc.c -I.**	
inserendo invece i file e il comando in un Makefile in questo modo: Makefile 1 CC=cc CFLAGS=-I. hellomake: hellomake.o hellofunc.o $(CC) -o hellomake hellomake.o hellofunc.o eseguendo il comando make, questo senza argomenti, esegue la prima regola del file. NB: Prima di ogni gcc nel makefile ci deve essere un tab...	
manca una cosa: la dipendenza dai file di inclusione. Se dovessi apportare una modifica a hellomake.h, ad esempio, make non ricompilerebbe i file .c, anche se necessario. dobbiamo dire a make che tutti i file .c dipendono da determinati file .h. Possiamo farlo scrivendo una semplice regola e aggiungendola al makefile. Makefile 2 CC=gcc CFLAGS=-I. DEPS = hellomake.h %.o: %.c $(DEPS) $(CC) -c -o $@ $< $(CFLAGS) hellomake: hellomake.o hellofunc.o $(CC) -o hellomake hellomake.o hellofunc.o	forbidden
Questa aggiunta crea prima la macro DEPS, che è l'insieme di file .h da cui dipendono i file .c. Quindi definiamo una regola che si applica a tutti i file che terminano con il suffisso .o. **La regola dice che il file .o dipende dalla versione .c del file e dai file .h inclusi nella macro DEPS. La regola poi dice che per generare il file .o, make deve compilare il file .c usando il compilatore definito nella macro CC. Il flag -c dice di generare il file oggetto,**	

il -o $@ dice di mettere l'output della compilazione nel file denominato sul lato sinistro di : , il $< è il primo elemento nell'elenco delle dipendenze e il La macro CFLAGS è definita come sopra.

Come ultima semplificazione, utilizziamo le macro speciali **$@ e $^** , che sono rispettivamente i lati sinistro e destro di : , per rendere più generale la regola di compilazione generale.

Nell'esempio seguente, tutti i file di inclusione devono essere elencati come parte della macro DEPS e tutti i file oggetto devono essere elencati come parte della macro OBJ.

Makefile 3

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h
OBJ = hellomake.o hellofunc.o

%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

hellomake: $(OBJ)
	$(CC) -o $@ $^ $(CFLAGS)
```

Quindi cosa succede se vogliamo iniziare a mettere i nostri file .h in una directory include, il nostro codice sorgente in una directory src e alcune librerie locali in una directory lib? Inoltre, possiamo in qualche modo nascondere quei fastidiosi file .o che si trovano dappertutto? La risposta, ovviamente, è sì. Il seguente makefile definisce i percorsi delle directory include e lib e inserisce i file oggetto in una sottodirectory obj all'interno della directory src. Ha anche una macro definita per tutte le librerie che vuoi includere, come la math library -lm . Questo makefile dovrebbe trovarsi nella directory src. Si noti che include anche una regola per ripulire le directory dei sorgenti e degli oggetti se si digita make clean . La regola .PHONY impedisce a make di fare qualcosa con un file chiamato clean.

Esempio di Makefile

```
NAME = libft.a

CC = gcc

AR = ar -rcs

FLAG = -Werror -Wall -Wextra

SRC = ft_atoi.c ft_putchar_fd.c ft_strjoin.c ft_strtrim.c\
      ft_bzero.c ft_putendl_fd.c ft_strlcat.c ft_substr.c\
      ft_calloc.c ft_putnbr_fd.c ft_strlcpy.c ft_tolower.c\
      ft_isalnum.c ft_memchr.c ft_putstr_fd.c ft_strlen.c\
      ft_toupper.c ft_isalpha.c ft_memcmp.c ft_split.c\
      ft_strmapi.c ft_isascii.c ft_memcpy.c ft_strchr.c\
      ft_strncmp.c ft_isdigit.c ft_memmove.c ft_strdup.c\
      ft_strnstr.c ft_isprint.c ft_memset.c ft_striteri.c\
```

```
	ft_strrchr.c ft_itoa.c

SRC_BONUS = ft_lstadd_front.c ft_lstnew.c ft_lstsize.c ft_lstadd_back.c\
		ft_lstlast.c ft_lstdelone.c ft_lstclear.c ft_lstiter.c\
		ft_lstmap.c

OBJ = $(SRC:.c=.o)

OBJ_BONUS = $(SRC_BONUS:.c=.o)

all: $(NAME)

$(NAME): $(OBJ)
	$(AR) $(NAME) $(OBJ)

%.o: %.c
	$(CC) -c $(FLAG) -I. $< -o $@

clean:
	rm -f $(OBJ) $(OBJ_BONUS)

fclean: clean
	rm -f $(NAME)

re: fclean all

bonus:  $(OBJ_BONUS)
	$(AR) $(NAME) $(OBJ_BONUS)

.PHONY: bonus all clean fclean re
```

```
Alcune MACRO

# \ = is used to Splitting Long Lines 3.1.1

# ;\ = indicates a multiline command and keeps the instance of the
terminal for
# the next command

# % = the same as * 'wildcard'
# $@ = means what is before the : in the target
# $^ = means what is after the : in the target
# $< = the first prerequisite (usually a source file)
# -I.  = adds include directory of header files.
# -f = force the removal even if the files have been already deleted.

# -c = Compile or assemble the source files, but do not link.
# The linking stage simply is not done. The ultimate output is
# in the form of an object file for each source file.
# By default, the object file name for a source file is made by replacing
# the suffix .c, .i, .s, etc., with .o. Unrecognized input files,
# not requiring compilation or assembly, are ignored.
```