Makefile

Un makefile consiste di alcune regole cosi' descritte:

```
TARGET: DIPENDENZE
        COMANDO
```

Se le regole sono memorizzate in un file chiamato Makefile o makefile e' sufficiente digitare il comando make seguito dal target che si vuole aggiornare (altrimenti si deve usare l'opzione -f per specificare il file corretto).Se non si specifica alcun target, viene eseguito automaticamente il primo.

Makefile

Di solito TARGET e' il nome dell'eseguibile o del file oggetto che si vuole generare, ma puo' anche essere un'azione ad esempio clean, una sorta di identificativo dell'azione da eseguire, in tal caso alla chiamata:

$ make clean

verra' eseguito il target “clean”

Makefile

Dipendenze, quando eseguire il comando ?

```
eseguibile: object1.o main.o
        comando
```

a sinistra dei due punti troviamo quindi il target, a destra troviamo la lista delle dipendenze che sono necessarie in qualche modo al target. Quando si esegue

```
$ make eseguibile
```

make controlla la data di ciascun oggetto, se questa e' piu' recente dell'eseguibile, allora viene eseguito il comando.

Makefile

Una caratteristica del make e' che le dipendenze (o “sorgenti”) del target sono “costruite” prima del confronto dei “timestamps”. In pratica la linea:

eseguibile: main.o

implica un “make main.o”:

main.o: main.c

comando1

che ha come “sorgente” main.c. Se main.c e' piu' recente di main.o, quest'ultimo viene ricostruito (cioe' viene seguito comando1). A questo punto main.o sara' piu' recente di eseguibile e quindi ?

Makefile

Vediamo e discutiamo qualche semplice Makefile (**Makefile1 Makefile2 Makefile3**)

Esercizio

Dati i files main1.c, sub1.c, sub2.c e sub1.h scrivere il Makefile corrispondente (**Makefile**).

Inter-language communication

“Mescolare” files scritti in linguaggi differenti e' relativamente facile nel caso ad esempio di unn “mixing” tra linguaggio C e C++ (di fatto il C e' un “subset” del C++). Nel caso in cui invece sia necessario utilizzare altri linguaggi, l'operazione puo' risultare maggiormente “complicata”.

Noi prenderemo in cunsiderazione il “mixing” tra tre linguaggi C, C++ e Fortran77, e lo faremo mediante qualche semplice esempio, cosi' da mostrare alcuni degli aspetti salienti realti all'Inter-language communication.

Passaggio Parametri ad una funzione (wikipedia)

Passaggio per valore Il **valore** dei parametri attuali viene copiato nelle variabili della funzione chiamata che rappresentano i parametri formali. Se la funzione chiamata li modifica, la funzione chiamante non potra' vedere queste modifiche. Si tratta quindi di passaggio unidirezionale

Passaggio per indirizzo o per riferimento La funzione invocata riceve un puntatore o riferimento ai parametri attuali. Se modifica un parametro passato per indirizzo, la modifica sara' visibile anche alla funzione chiamante. Il passaggio e' quindi potenzialmente bidirezionale.

Allocazione dinamica F77

In fortran77 il passaggio dei parametri alla funzione avviene per riferimento, non esistendo di fatto il tipo puntatore. Di fatto noi per “contenere” il puntatore restituito da malloc useremo un “integer*4” o “integer*8” a seconda dell'architettura dell'elaboratore (vedi main.F). E' possibile tuttavia usare l'operatore %val() almeno nel caso del compilatore g77.

L'altra particolarita' e' che come e' possibile vedere il compilatore aggiunge un “_” al nome delle funzioni.

Altre particolarita' e caratteristiche solo cenni.

Allocazione dinamica F77

```
$ g77 -c -o main.o main.F
...
$ nm main.o
0000000000000000 T MAIN__
0000000000000000 d __g77_cilist_0.0
                 U do_lio
                 U e_rsle
0000000000000091 T fillmat_
                 U free_
                 U malloc_
                 U s_rsle
                 U s_stop
```

Allocazione dinamica F77

util.c

```
#include <stdlib.h>
void * malloc_ (int * size) {
  return malloc ((*size));
}

void free_ (void ** ptr) {
  free ((*ptr));
  return;
}
```

Allocazione dinamica F77

```
$ g77 -o main main.o util.o
$ g77 -v -o main main.o util.o
```

Usaremo il g77 per linkare. Con l'opzione -v possiamo visualizzare tutte le librerie di runtime fortran che vengono incluse durante il linking. Chiaramente posso usare anche il gcc per eseguire il linking:

```
$ gcc -o main main.o util.o -L/usr/lib/gcc/x86_64-redhat-linux/3.4.6/ -lg2c -lfrtbegin -lm
```

C/C++

L'overloading delle funzioni e' una funzionalita' specifica del C++ che non e' presente in C. Questa funzionalita' permette di poter utilizzare lo stesso nome per una funzione piu' volte all'interno dello stesso programma, a patto pero' che gli argomenti forniti siano differenti.

La comprensione di questo meccanismo e' strettamente correlata alla comprensione del meccanismo alla base dell'inter-laganuage communication tra C e C++.

Vediamo ad esempio il sorgente main.cpp, dove sono definite due funzioni con stesso nome ma diversi parametri.

Overloading C++

```
void sub (int a) {
  return;
}

void sub (float a) {
  return;
}

int main (int argc, char ** argv) {
  int a;
  float b;
  sub (a);
  sub (b);
  return 0;
}
```

Overloading C++

```
$ gcc -c -o main.o main.cpp
```

```
$ nm main.o
000000000000000a T _Z3subf
0000000000000000 T _Z3subi
                 U __gxx_personality_v0
0000000000000016 T main
```

I simboli relati alle due funzioni differiscono per la lettera finale: “f” nel caso della funzione che prende come parametro un “float” ed “i” nell'altro caso (i.e. “int”)

C/C++

Chiamata a funzione C++ da sorgenti C (vedi directory **3**):

```
#ifndef _UTIL_INC
#define _UTIL_INC
#ifdef __cplusplus
extern "C" {
#endif
    void ins (int);
    void print ();
#ifdef __cplusplus
}
#endif
#define PIPPO 9
#endif
```

C/C++

```
$ make
gcc -O0 -W -Wall -I./ -O0 -W -Wall -I./  -c -o main.o main.c
g++  -O0 -W -Wall -I./  -c -o util.o util.cpp
gcc -o test main.o util.o -lstdc++
$ nm util.o | grep "ins"
0000000000000072 t _GLOBAL__I_ins
0000000000000000                                                 W
_ZNSt6vectorIiSaIiEE13_M_insert_auxEN9__gnu_cxx17__normal_iteratorIPiS1_EERKi
0000000000000122 T ins
```

In C++, una keyword indica che una funzione deve impiegare la convenzione di linking (il termine comunemente usato e' "binding" - N.d.T.) usata in C: `extern "C"`. Una funzione dichiarata `extern "C"` viene rappresentata da un simbolo coincidente con il nome della funzione, esattamente come in C. Per questa ragione, solo funzioni che non sono metodi di una classe possono essere dichiarate `extern "C"`, e non possono essere overloaded.

Esercizio

Da quel poco che evete visto del Fortran 77 dovreste essere in grado di scrivere un programma in Fortran 77 che faccia la stessa cosa del main.c dell'esempio precedente (directory 3 per intenderci), cioe' usi le funzioni C++ ins e print di util.cpp. Usate il compilatore gfortran e non il g77. (**soluzione**)