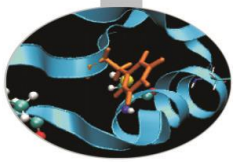
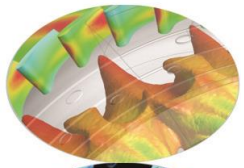
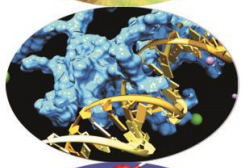
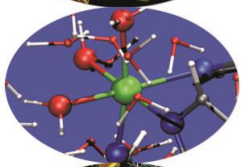
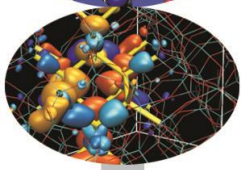
CINECA SCAI
SuperComputing Applications and Innovation



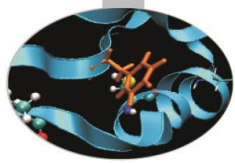
Preprocessore e Compilazione

P. Dagna, *CINECA*

2017

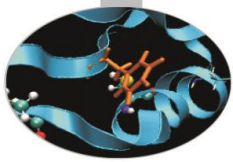





CINECA



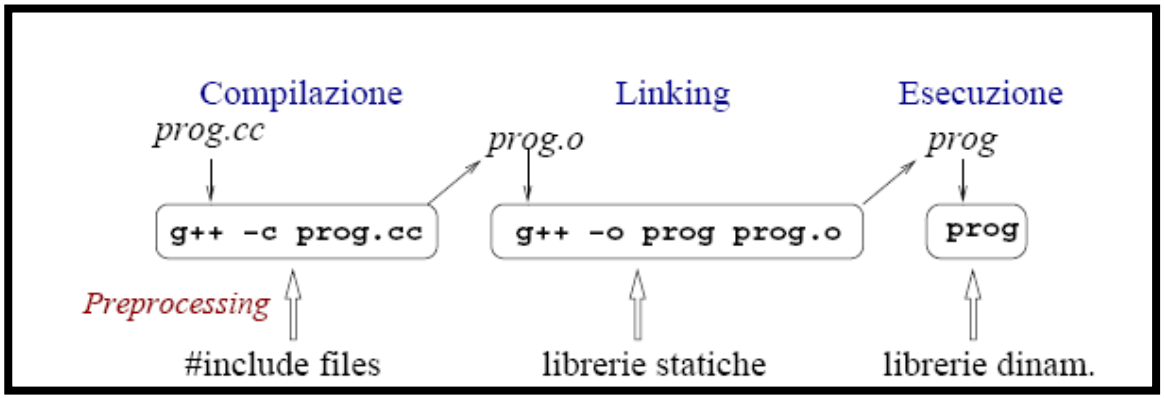
Indice

- Il preprocessore
- Le istruzioni per il preprocessore
- Le MACRO
- L'ambiente linux: alcuni comandi
- Editor ed il compilatore gcc

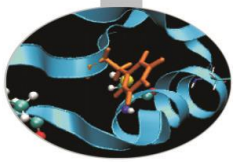


Il preprocessore

Il processo di compilazione C/C++ comprende diverse fasi:



- La prima fase viene detta di istruzioni al pre-processore e prevede ad esempio l'inclusione di librerie che contengono le definizioni di funzioni usate nel programma.
- Le direttive al pre-processore sono numerose, e si distinguono nel codice per la presenza del simbolo # all'inizio della dichiarazione; noi consideremo solo le principali:
- **#include, #define, #if, #ifndef, #ifdef, #else, #elif, #endif,#undef ; insieme con gli operatori: #, ##**



#include

- Questa direttiva forza il compilatore a leggere e compilare un altro codice sorgente
- Ne esistono due forme a livello di sintassi:

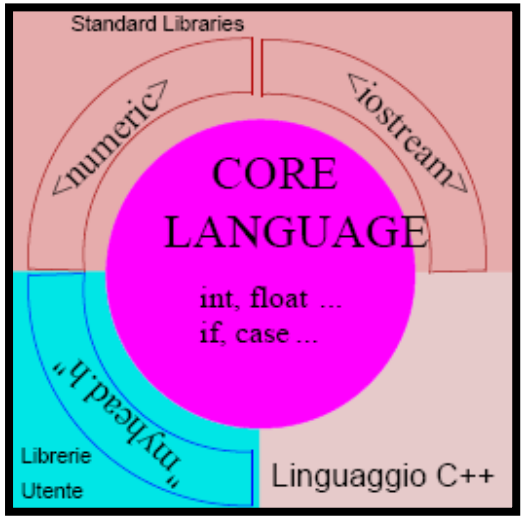
  #include”nome-file”

  #include<nome-file>

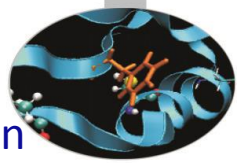
Distinti convenzionalmente in base la fatto che il file che si vuole includere sia parte dell’installazione standard del compilatore (<>) oppure che sia stato creato dall’utente (“”).

Esempio:

```
#include<stdio.h>

#include”mia_lib.h”
```



#define



- Questa direttiva è utilizzata per effettuare macro sostituzioni di parti di codice con altre.
- La forma generale è data da:

  #define nome-macro sequenza-di-caratteri

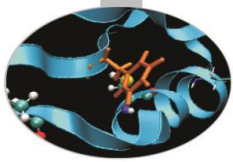
  /*ad ogni occorrenza nel codice del nome-macro viene sostituita la sequenza-di-caratteri*/
- La macro può avere argomenti e in questo caso la macro assomiglia molto ad una funzione.

Esempio:

```
#include<stdio.h>
#define MYMACRO(a) ((a)<0 ? -(a) : (a))

int main(){
printf("valore assoluto di -8: %d \n",MYMACRO(-8));
return 0;
}
```

NOTA: L'introduzione dell'istruzione "inline" e del qualificatore "const" a partire dal C99 ha di fatto permesso di soppiantare l'uso delle macro definite tramite la direttiva #define tipiche dello standard C89.

#if #ifdef #ifndef #else #elif #endif

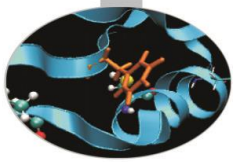


- Questo insieme di direttive è utilizzata per imporre al compilatore delle compilazioni selettive di varie porzioni di codice.
- L'idea generale è la seguente: se l'espressione a valle del controllo tramite le istruzioni **#if, #ifdef o #ifndef** è vera allora il codice compreso tra quel punto e la direttiva di termine (**#endif)** verrà compilato altrimenti verrà saltato.
- L'uso della direttiva #else è intuitivo.

Esempio:

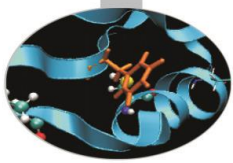
```
#ifndef nome-macro
#define nome-macro
//…
#else …
#endif
//
```

#if #ifdef #ifndef #else #elif #endif



- Tipico uso ne viene fatto nella scrittura di proprie librerie per evitare di ridefinire le stesse variabili qualora la libreria venisse richiamata da più file nello stesso programma.

```
//nel file header.h
#ifndef header_h_
#define header_h_
  //codice
#endif
```



#undef

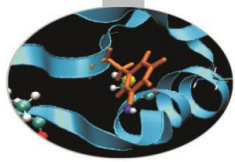
- Questa direttiva è utilizzata per effettuare la rimozione di macro definite precedentemente
- La forma generale è data da:

#undef nome-macro

Esempio:

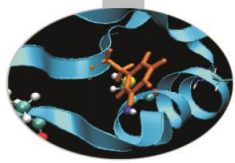
```
#define MYLEN 1000
#define MYWIDTH 1000
char array[MYLEN][MYWIDTH];
#undef MYLEN

/* la macro MYLEN non esiste più, mentre MYWIDTH
   continua ad essere definita */
```

Gli operatori # e



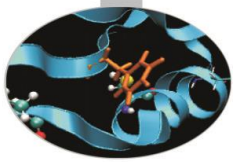
- Questi operatori vengono utilizzati all'interno di una #define macro.
- L'operatore # posizionato in una #define macro impone che l'argomento che lo segue sia trasformato in una stringa.
- L'operatore ## è usato per concatenare, in una #define macro, due argomenti:

Gli operatori # e



- Esempio di codice C che interpreta comandi

```
struct command
      {
        char *name;
        void (*function) (void);
      };
      struct command commands[] =
      {
        { "quit", quit_command },
        { "help", help_command }
      };
#define COMMAND(NAME)  { #NAME, NAME ## _command }
      struct command commands[] =
      {
        COMMAND (quit),
        COMMAND (help)
      };
```



Compilazione ed esecuzione

- Per compilare semplici programmi, costituiti da pochi files sorgenti, in generale si utilizza direttamente il compilatore nel modo seguente:
  - Generazione eseguibile dalla riga di comando della shell di lavoro tramite i comandi:

    user@linux>gcc file.c

    oppure

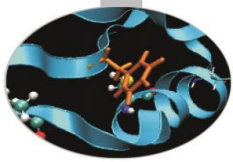
    user@linux>gcc –o eseguibile.exe file.c

  - Esecuzione codice compilato:

    user@linux>./a.out

    oppure

    user@linux>./eseguibile.exe



Compilazione ed esecuzione

Supponendo di disporre di di un programma costituito da:

- mioprog.h file dichiarativo
- mioprog.c file di definizioni
- main.c file di utilizzo

le istruzioni per la compilazione del sorgente e la creazione dell'eseguibile sono:

1- Compilazione dei sorgenti:

gcc -c mioprog.c

gcc -c main.c

che produce come output i file oggetto *mioprog.o* e *main.o*

2- Linking:

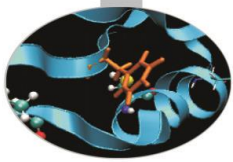
gcc -o mioexe mioprog.o main.o

Oppure unendo compilazione e linking:

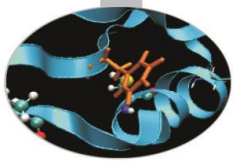
gcc -o mioexe mioprog.c main.c

Le dipendenze tra i file e le specifiche istruzioni di compilazioni vengono solitamente gestite attraverso un Makefile.

Ambienti Linux e codici C



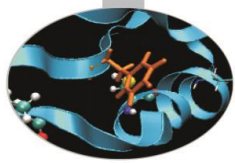
- Il pc a vostra disposizione fa uso di linux
- Aprendo una shell di lavoro si può creare ed accedere ad uno spazio di lavoro personale tramite i comandi:
- **user@linux> mkdir** *"nome della cartella"*
- **user@linux> cd** *"nome della cartella"*
- in questo spazio inizialmente non vi saranno file:
- **user@linux> ls –la [–tr –h]**



Ambienti Linux e codici C

Comando	**Azione**
ls –la [-tr –h]	listato ordinato per data dei file contenuti nella cartella di lavoro
mkdir “myworkdir”	creazione di una cartella di lavoro
cd /path/myworkdir	cambio di cartella di lavoro dalla posizione corrente a “myworkdir”
rm “nome_file”	rimozione del file “nome_file”

Editing



- Gli editor di testo disponibili in ambiente Linux sono:
  - **nedit** (più semplice ed intuitivo)
  - **emacs** (decisamente più potente e complesso)
  - **Code::Blocks** (ambiente di sviluppo comprensivo di GUI)
- In ambiente Windows
  - **Dev-C++** (http://www.bloodshed.net/dev/)
  - **Code::Blocks** (http://www.codeblocks.org/)
- Questi editor riconoscono la sintassi C/C++ e agevolano la lettura del codice grazie all'uso dei colori.