4) Primi elementi di programmazione in C e creazione di un programma eseguibile

Livelli di programmazione

- Si può programmare a diversi livelli utilizzando linguaggi caratterizzati da diversi livelli di astrazione: più basso è il livello più vicino alla macchina (e ai suoi dettagli costruttivi) è il linguaggio.

- Un programma scritto utilizzando il linguaggio proprio di un livello di programmazione è destinato ad essere tradotto (da un programma traduttore) in un linguaggio di livello inferiore.
- Se la traduzione produce un programma eseguibile successivamente riutilizzabile si parla di *compilazione* del codice.
- Del codice tradotto in maniera "simultanea" si dice invece *interpretato*.

Linguaggi di alto livello

- I vantaggi della programmazione in un linguaggio di alto livello sono
  - l'astrazione: indipendenza della descrizione dell'algoritmo dai dettagli della macchina che lo deve eseguire;
  - la portabilità del codice: sottoprodotto dell'astrazione che consente l'esecuzione dello stesso programma su una vasta gamma di calcolatori diversi, a condizione che esista un compilatore in grado di tradurlo nel linguaggio di ogni macchina.

Sviluppo del *software*

Due diverse filosofie di programmazione:

- Procedurale: si costruisce il diagramma di flusso delle informazioni e lo si traduce in una sequenza di istruzioni, ovvero si agisce in modo sequenziale su insiemi di dati modificandoli. Il programma consiste in una successione di azioni sui dati che sono soggetti passivi.

- Orientata agli oggetti: ci si concentra sulla identificazione degli oggetti che si vogliono manipolare, sulle loro proprietà e sulle relazioni tra di essi. Per ogni tipo di oggetto si scrive (o si trova già scritto) il codice che ne consente la manipolazione, il programma vero e proprio si limita a creare gli oggetti e a lasciarli interagire.

Linguaggi di programmazione

- Pascal, Fortran e C sono alcuni dei linguaggi tradizionalmente usati in ambito scientifico e sono linguaggi procedurali.

- La programmazione orientata agli oggetti ha sviluppato dei linguaggi che sono più idonei a questa filosofia e che forniscono gli strumenti per gestire gli oggetti, primi fra tutti JAVA e C++.

- In questo corso abbiamo scelto di adottare il linguaggio C per introdurre gli elementi base necessari alla programmazione scientifica.

sviluppo di un programma

la scrittura di un programma, anche semplice, comporta

- analisi del problema, individuazione delle quantita' in ingresso ed in uscita, scelta dell'algoritmo, individuazione di eventuali componenti gia' sviluppate e riutilizzabili, pianificazione delle procedure di collaudo
- scrittura del codice sorgente mediante un editor di testo (emacs ad esempio)
- produzione del codice eseguibile (compilazione) e risoluzione degli eventuali problemi
- collaudo:
  - il programma deve girare senza intoppi
  - il programma deve fornire risultati plausibili

  NB i test rivelano solo la presenza di errori ma non possono certificarne l'assenza!

Codice
C

main

- ogni applicazione software (programma) ha bisogno di un nucleo centrale che ne costituisce il punto di partenza, sia dal punto di vista logico che tecnico.
- in C questo nucleo centrale e' una funzione chiamata `main`.
- come tutte le funzioni C (lo vedremo in seguito) si tratta di un blocco di programma con la seguente struttura

```
tipo nome([argomenti]) {
...istruzioni...
}
```

- in questo caso la funzione non restituisce nulla e quindi il `tipo` non compare e il nome e' `main`

Esempio di codice in C

(dal testo Barone et al. listato 3.7)

$$T_C = (T_F - 32) \times 5/9$$

```
#include <stdio.h>
#define TF2TC

main() {
   double tc, tf, offset, conv;
   offset = 32.;

#ifdef TF2TC
   conv = 5. / 9.;
   printf("Valore in gradi Fahrenheit= ");
   scanf("%lf",&tf);
   tc = (tf - offset) * conv;
   printf("Valore in gradi celsius= %f\n",tc);
#endif

#ifndef TF2TC
   conv = 9. / 5.;
   printf("Valore in gradi celsius= ");
   scanf("%lf",&tc);
   tf = tc * conv + offset;
   printf("Valore in gradi Fahrenheit= %f\n",tf);
#endif
}
```

direttive del preprocessore

- nel listato del programma alcune righe iniziano con il simbolo #, sono istruzioni destinate ad un programma (preprocessore o precompilatore) che elabora il codice C prima della sua compilazione.
- le direttive sono usate per 3 scopi
  - inclusione di file contenenti dichiarazioni o definizioni

    ```
    #include <stdio.h>
    ```

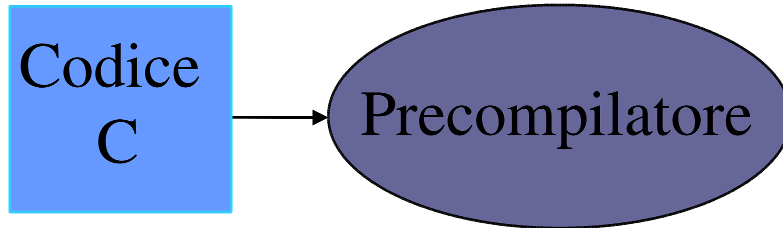
  - definizioni utili a ridurre le occorrenze nel codice di valori costanti

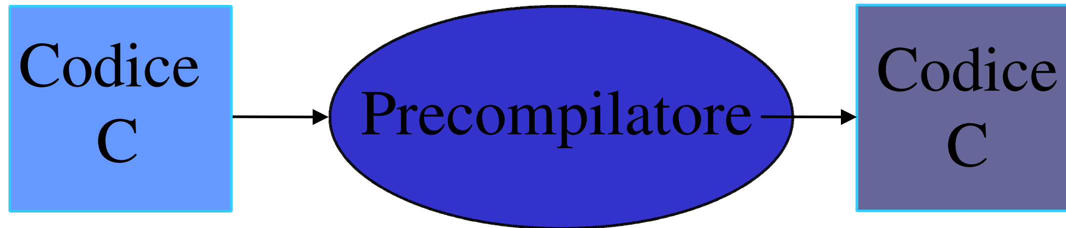
    ```
    #define MY_MAX 100
    ```

    o creazione di macro

    ```
    #define SUMQ(a,b) ((a*a)+(b*b))
    ```

  - scelta di parti di codice condizionata dalle direttive `#define`, `#undef`, `#if`, `#ifdef`, `#ifndef`, `#endif`


Codice C
Precompilatore


Codice
C
Precompilatore
Codice
C

Esempio di codice C precompilato (1)

La direttiva del preprocessore `#include <stdio.h>` provoca l'inserimento del contenuto del file `/usr/include/stdio.h` nel nostro file.
`#define TF2TC` seleziona il codice per la conversione da gradi Fahrenheit a Celsius.

```
/* circa 900 righe provenienti dall'inclusione di
stdio.h */

# 2 "cap03_7.c" 2
main() {
   double tc, tf, offset, conv;
   offset = 32.;

   conv = 5. / 9.;
   printf("Valore in gradi Fahrenheit= ");
   scanf("%lf",&tf);
   tc = (tf - offset) * conv;
   printf("Valore in gradi celsius= %f\n",tc);
# 23 "cap03_7.c"
}
```

Esempio di codice in C precompilato (2)

Rimuovendo la direttiva `#define TF2TC` viene selezionato il codice per la conversione da gradi Celsius a Fahrenheit.

```
/* circa 900 righe provenienti dall'inclusione di
stdio.h */

# 2 "cap03_7.c" 2

main() {
   double tc, tf, offset, conv;
   offset = 32.;
# 16 "cap03_7.c"
   conv = 9. / 5.;
   printf("Valore in gradi celsius= ");
   scanf("%lf",&tc);
   tf = tc * conv + offset;
   printf("Valore in gradi Fahrenheit= %f\n",tf);

}
```

istruzioni C

- le istruzioni del linguaggio C sono sequenze di simboli che il compilatore traduce in codice eseguibile
- il formato e' libero: le istruzioni non richiedono una formattazione specifica
- ogni singola istruzione termina con il carattere ;
- piu' istruzioni possono essere raggruppate a formare un'istruzione composta o blocco di codice medinate una coppia di { }
- un'istruzione puo' contenere varibili, costanti, operatori, parole chiave e funzioni
- una combinazione sintatticamente corretta di operatori, variabili e costanti si definisce espressione
- parti di codice racchiuse tra i simboli /* e */ contengono commenti e non vengono compilate.

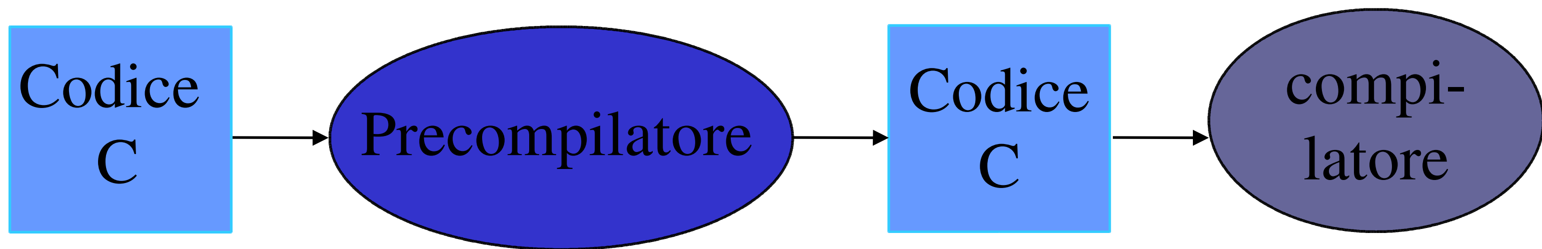
codice C

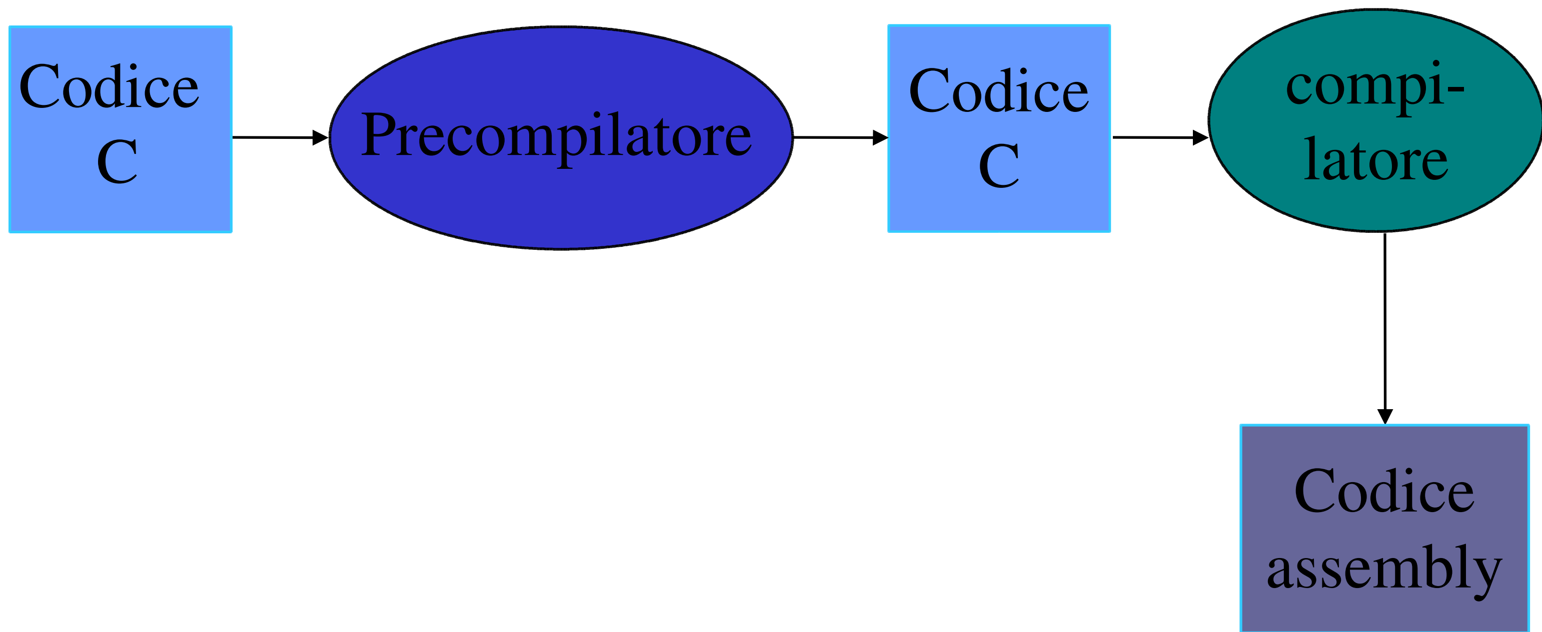
```
/* questo e' un commento:
   circa 900 righe provenienti dall'inclusione di stdio.h */

# 2 "cap03_7.c" 2   /* questa riga , prodotta dal preprocessore, non
                       viene vista dal compilatore */

main() {            /* qui inizia la funzione main */
   double tc, tf, offset, conv;  /* dichiarazione di 4 variabili
                                    di tipo double */
   offset = 32.;   /* inizializzazione della variabile offset*/
# 16 "cap03_7.c"   /* questa riga , prodotta dal preprocessore, non
                      viene vista dal compilatore */
   conv = 9. / 5.; /* inizializzazione della variabile conv*/
   printf("Valore in gradi celsius= "); /* chiamata di una funzione */
   scanf("%lf",&tc);          /* chiamata di una funzione:
                                 lettura del valore di input di tc */
   tf = tc * conv + offset;   /* calcolo di tf */
   printf("Valore in gradi Fahrenheit= %f\n",tf); /* chiamata di una
                                                     funzione: stampa
                                                     di tf */

} /* qui termina la funzione main */
```


Codice
C
Precompilatore
Codice
C
compi-
latore


Codice C
Precompilatore
Codice C
compi-
latore
Codice assembly

Esempio di codice assembly

```
        .file   "cap03_7.c"
        .section        .rodata
.LC0:
        .string "Valore in gradi celsius= "
.LC1:
        .string "%lf"
        .align 4
.LC2:
        .string "Valore in gradi Fahrenheit= %f\n"
        .text
.globl main
        .type   main,@function
main:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        andl    $-16, %esp
        movl    $0, %eax
        subl    %eax, %esp
        movl    $0, -24(%ebp)
        movl    $1077936128, -20(%ebp)
        movl    $-858993459, -32(%ebp)
        movl    $1073532108, -28(%ebp)
```

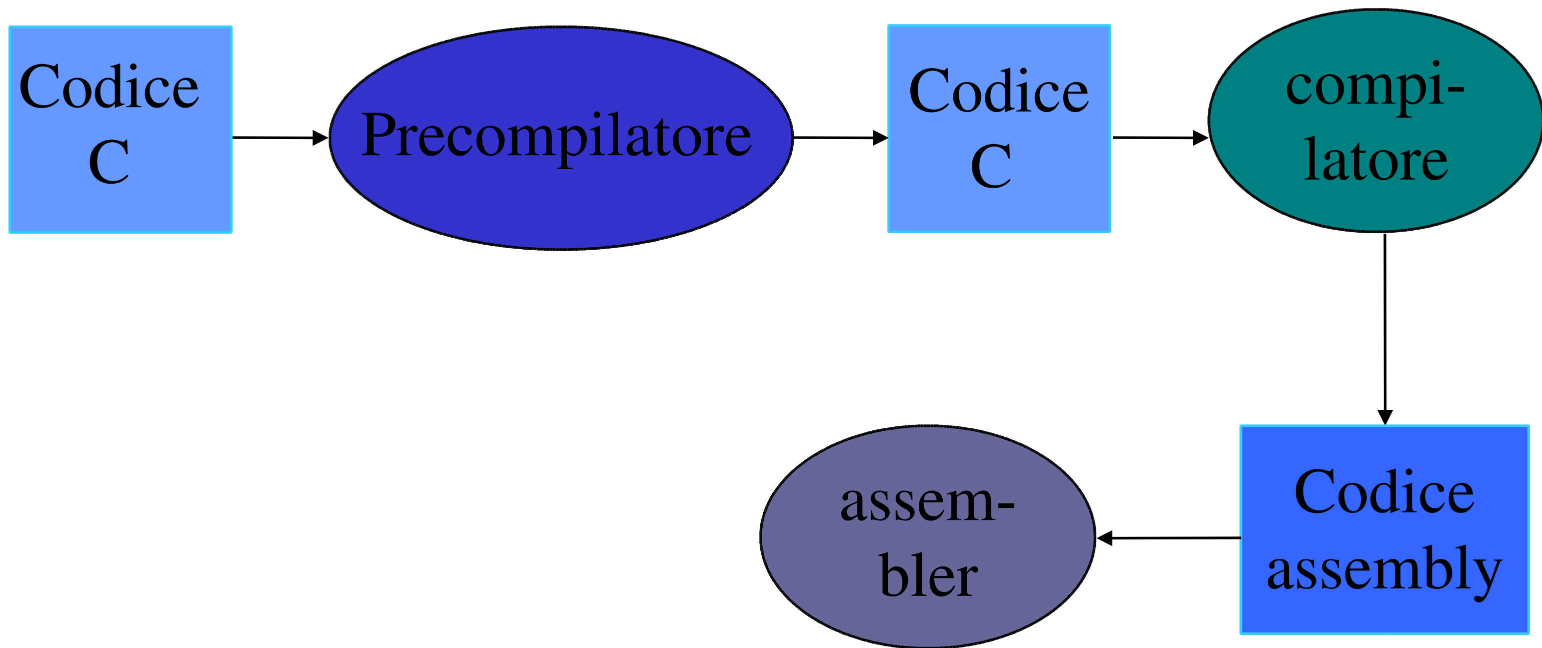
1

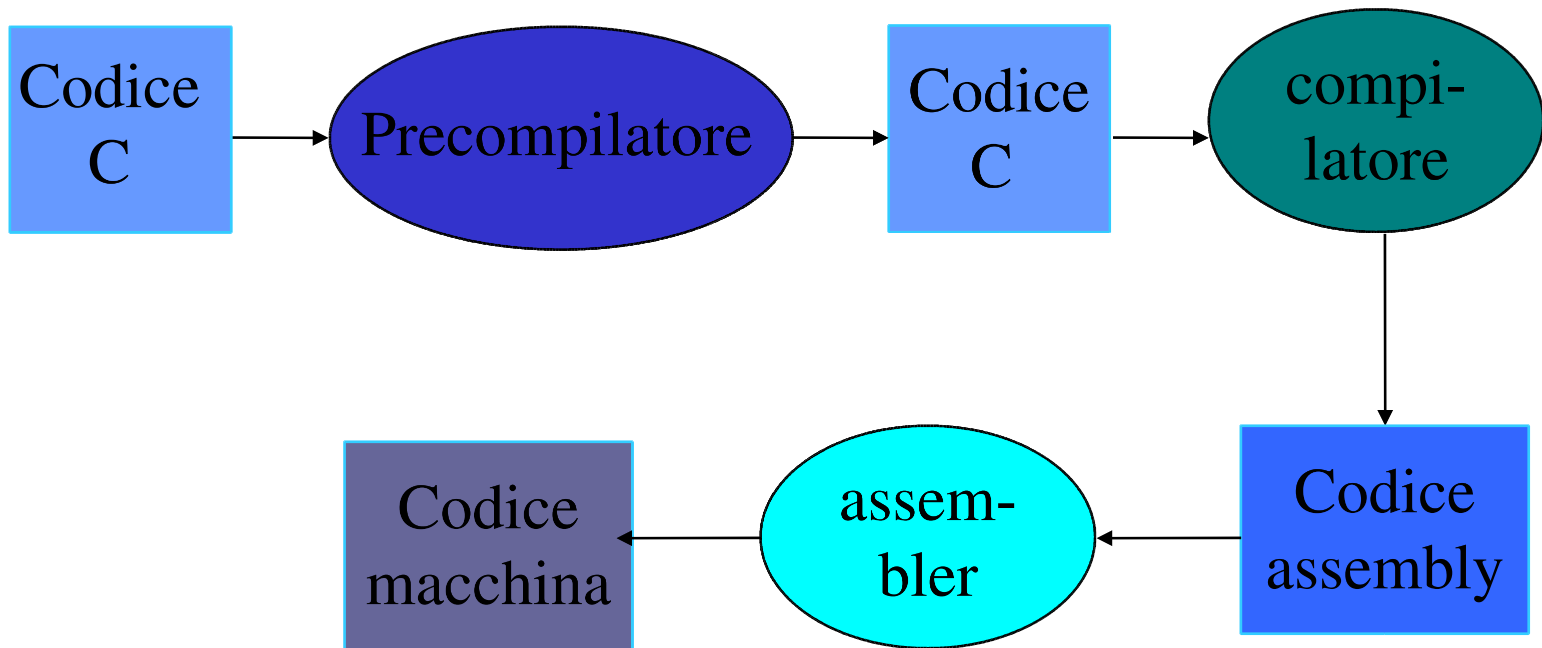
```
        movl    $-858993459, -32(%ebp)
        movl    $1073532108, -28(%ebp)
        subl    $12, %esp
        pushl   $.LC0
        call    printf
        addl    $16, %esp
        subl    $8, %esp
        leal    -8(%ebp), %eax
        pushl   %eax
        pushl   $.LC1
        call    scanf
        addl    $16, %esp
        fldl    -8(%ebp)
        fmull   -32(%ebp)
        faddl   -24(%ebp)
        fstpl   -16(%ebp)
        subl    $4, %esp
        pushl   -12(%ebp)
        pushl   -16(%ebp)
        pushl   $.LC2
        call    printf
        addl    $16, %esp
        leave
        ret
```

2

```
.Lfe1:
        .size   main,.Lfe1-main
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.2.3 20030502 (Red Hat Linux 3.2.3-56)"
~
```

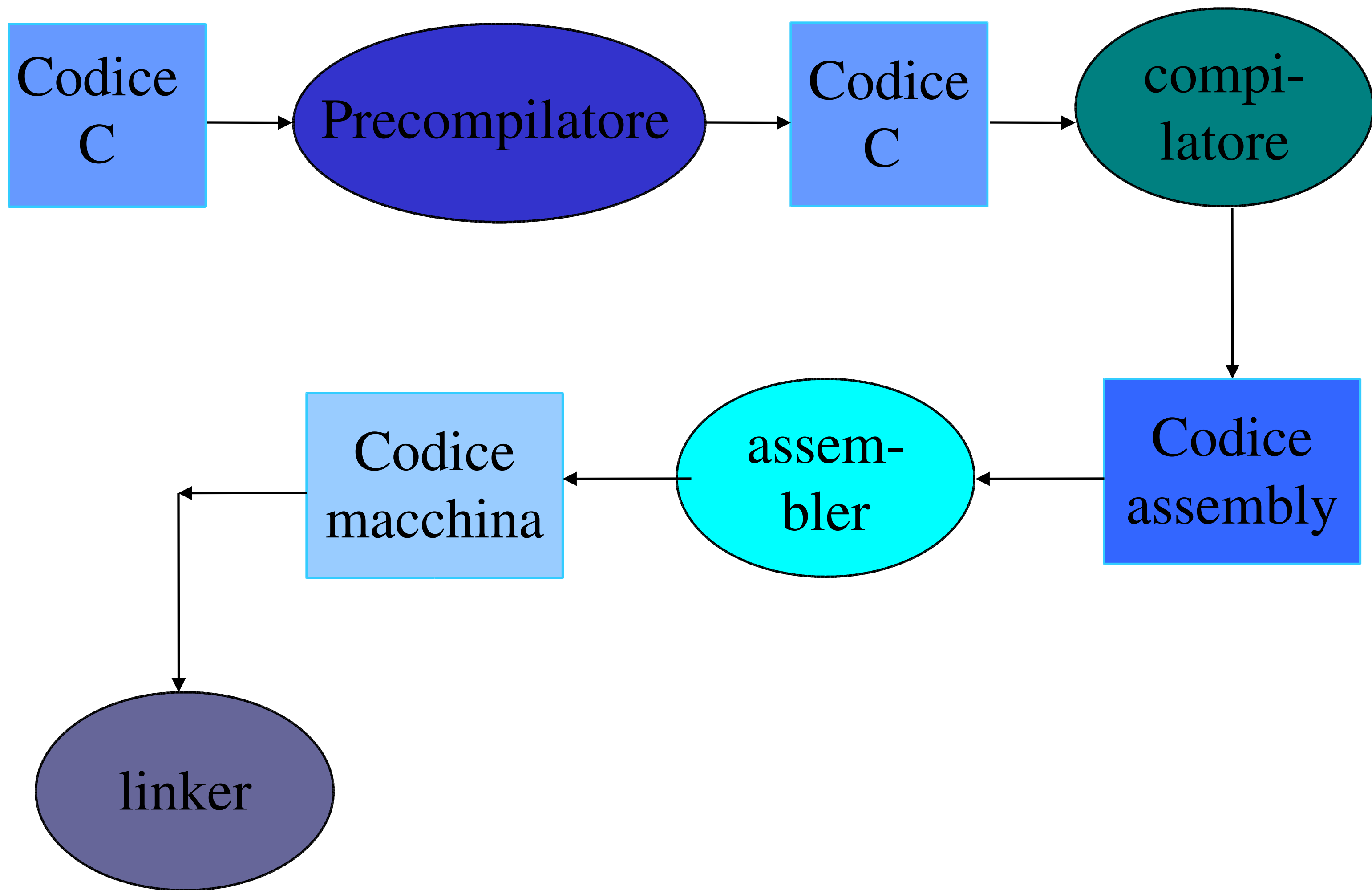
3

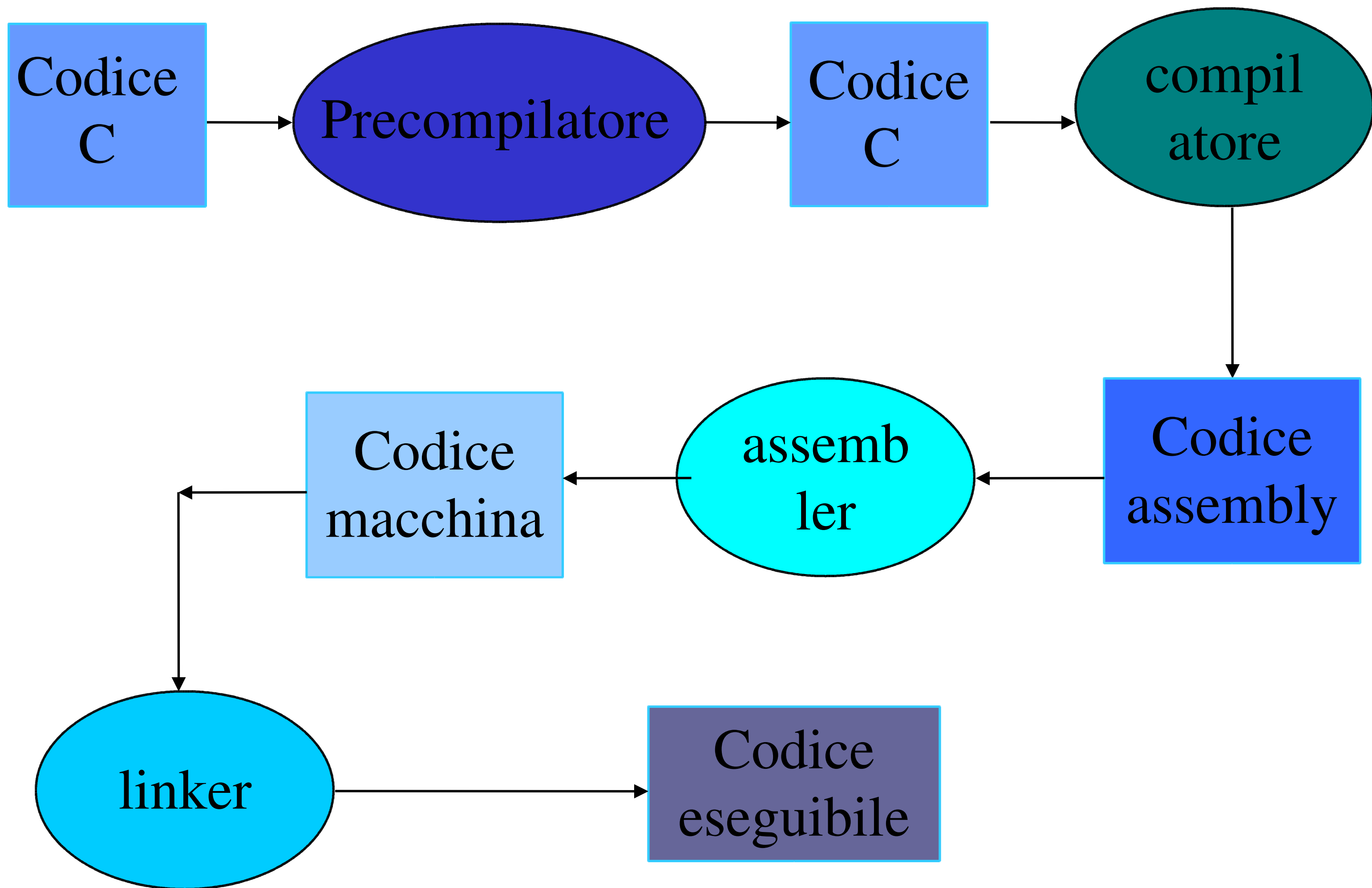

Codice
C
Precompilatore
Codice
C
compi-
latore
Codice
assembly
assem-
bler

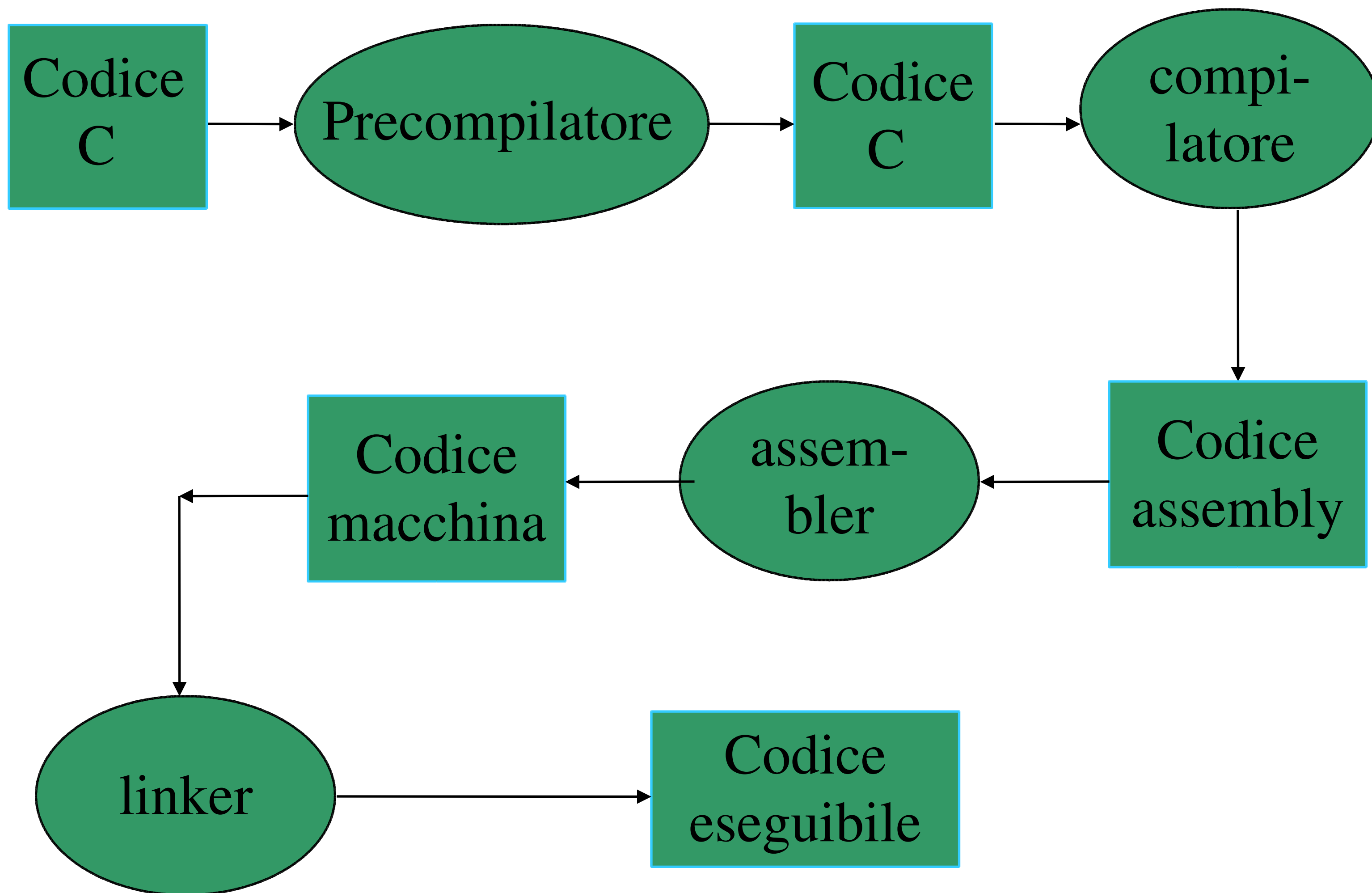

Codice
C
Precompilatore
Codice
C
compi-
latore
Codice
assembly
assem-
bler
Codice
macchina

Esempio di codice macchina

```
0000000 457f 464c 0101 0001 0000 0000 0000 0000 0001 0003 0001 0000 0000 0000 0000 0000
0000040 0170 0000 0000 0000 0034 0000 0000 0028 000b 0008 8955 83e5 28ec e483 b8f0 0000
0000100 0000 c429 45c7 00e8 0000 c700 ec45 0000 4040 45c7 cde0 cccc c7cc e445 cccc 3ffc
0000140 ec83 680c 0000 0000 fce8 ffff 83ff 10c4 ec83 8d08 f845 6850 001a 0000 fce8 ffff
0000200 83ff 10c4 45dd dcf8 e04d 45dc dde8 f05d ec83 ff04 f475 75ff 68f0 0020 0000 fce8
0000240 ffff 83ff 10c4 c3c9 6156 6f6c 6572 6920 206e 7267 6461 2069 6563 736c 7569 3d73
0000300 0020 6c25 0066 0000 6156 6f6c 6572 6920 206e 7267 6461 2069 6146 7268 6e65 6568
0000340 7469 203d 6625 000a 4700 4343 203a 4728 554e 2029 2e33 2e32 2033 3032 3330 3530
0000400 3230 2820 6552 2064 6148 2074 694c 756e 2078 2e33 2e32 2d33 3635 0029 2e00 7973
0000440 746d 6261 2e00 7473 7472 6261 2e00 6873 7473 7472 6261 2e00 6572 2e6c 6574 7478
0000500 2e00 6164 6174 2e00 7362 0073 722e 646f 7461 0061 6e2e 746f 2e65 4e47 2d55 7473
0000540 6361 006b 632e 6d6f 656d 746e 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000600 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 001f 0000 0001 0000
0000640 0006 0000 0000 0000 0034 0000 0074 0000 0000 0000 0000 0000 0004 0000 0000 0000
0000700 001b 0000 0009 0000 0000 0000 0000 0000 03f8 0000 0030 0000 0009 0000 0001 0000
0000740 0004 0000 0008 0000 0025 0000 0001 0000 0003 0000 0000 0000 00a8 0000 0000 0000
0001000 0000 0000 0000 0000 0004 0000 0000 0000 002b 0000 0008 0000 0003 0000 0000 0000
0001040 00a8 0000 0000 0000 0000 0000 0000 0000 0004 0000 0000 0000 0030 0000 0001 0000
0001100 0002 0000 0000 0000 00a8 0000 0040 0000 0000 0000 0000 0000 0004 0000 0000 0000
0001140 0038 0000 0007 0000 0000 0000 0000 0000 00e8 0000 0000 0000 0000 0000 0000 0000
0001200 0001 0000 0000 0000 0048 0000 0001 0000 0000 0000 0000 0000 00e8 0000 0034 0000
0001240 0000 0000 0000 0000 0001 0000 0000 0000 0011 0000 0003 0000 0000 0000 0000 0000
0001300 011c 0000 0051 0000 0000 0000 0000 0000 0001 0000 0000 0000 0001 0000 0002 0000
0001340 0000 0000 0000 0000 0328 0000 00b0 0000 000a 0000 0008 0000 0004 0000 0010 0000
0001400 0009 0000 0003 0000 0000 0000 0000 0000 03d8 0000 001d 0000 0000 0000 0000 0000
0001440 0001 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 0000 0000 0000
0001500 0000 0000 0004 fff1 0000 0000 0000 0000 0000 0000 0003 0001 0000 0000 0000 0000
0001540 0000 0000 0003 0003 0000 0000 0000 0000 0000 0000 0003 0004 0000 0000 0000 0000
0001600 0000 0000 0003 0005 0000 0000 0000 0000 0000 0000 0003 0006 0000 0000 0000 0000
0001640 0000 0000 0003 0007 000b 0000 0000 0000 0074 0000 0012 0001 0010 0000 0000 0000
0001700 0000 0000 0010 0000 0017 0000 0000 0000 0000 0000 0010 0000 6300 7061 3330 375f
0001740 632e 6d00 6961 006e 7270 6e69 6674 7300 6163 666e 0000 0000 0030 0000 0501 0000
0002000 0035 0000 0902 0000 0044 0000 0501 0000 0049 0000 0a02 0000 0066 0000 0501 0000
0002040 006b 0000 0902 0000
```


Codice
C
Precompilatore
Codice
C
compi-
latore
Codice
assembly
assem-
bler
Codice
macchina
linker


Codice
C
Precompilatore
Codice
C
compil
atore
Codice
assembly
assemb
ler
Codice
macchina
linker
Codice
eseguibile


Codice
C
Precompilatore
Codice
C
compi-
latore
Codice
assembly
assem-
bler
Codice
macchina
linker
Codice
eseguibile

Linker

- Riprendendo il nostro esempio che contiene delle istruzioni di I/O (input/output). La direttiva del precompilatore `#include <stdio.h>` ci ha permesso di includere le dichiarazioni delle funzioni `printf` e `scanf` che usiamo per scrivere su schermo e leggere da tastiera. Senza queste dichiarazioni avremmo potuto avere degli errori di compilazione.
- Ma dove si trova il codice di queste funzioni? Nella libreria (dall'inglese library: biblioteca) del C che contiene il codice macchina per tutti gli oggetti predefiniti.
- Il linker di `gcc` (ma non `ld`!) cerca automaticamente il codice riferito dall'utente (e non presente nel codice sorgente) nelle librerie di sistema. Qualora non venisse trovata la funzione richiesta si avrebbe un errore del linker. Ad esempio usando `ld`

```
cap03_7.o(.text+0x35): In function `main':
: undefined reference to `printf'
```

Linker

- E se si volesse usare del codice non di sistema? Per esempio dell'altro codice già scritto dall'utente?

- Il linker accetta in input più di un file e quindi è sufficiente passagli oltre al programma utente anche gli altri file o librerie contenenti il codice che vuole riutilizzare.

Estensione dei file

- Nella maggior parte dei sistemi operativi il nome dei file è costituito da due parti separate da un punto:

  nome.estensione

- Il nome può essere un qualunque identificatore valido, l'estensione in genere identifica il tipo di file e i programmi che accettano file in input generalmente controllano l'estensione per verificarne il tipo.

- Le estensioni dipendono dal sistema operativo e dai programmi applicativi.

- In questo corso useremo l'estensione .c per i file che contengono codice sorgente C (estensioni valide sono anche C e cc).

Uso di gcc (compilatore GNU)

- `gcc esempio.c`

  applica precompilatore, compilatore, assembler e linker e produce l'eseguibile `a.out`

- `gcc esempio.c -o esempio`

  chiama l'eseguibile prodotto `esempio anziche' a.out`

  attenzione! il blocco istruzione-argomento -o esempio puo' essere messo anche subito dopo il comando gcc ma in tal caso, qualora si dimenticasse l'argomento esempio si sovrascriverebbe il proprio codice!

- `gcc esempio.o altrofile.o libreria.a`

  invoca solo il linker, cercando i riferimenti esterni in `altrofile.o` e in `libreria.a`

- `gcc -DVARPREPR esempio.c`

  definisce esternamente la variabile VARPREPR del preprocessore

Uso di gcc (compilatore GNU)

- Possiamo anche effettuare i singoli passaggi intermedi usando le opzioni di gcc (consultabili con `man gcc`)
- `gcc -E esempio.c`

  invoca solo il precompilatore (e non salva l'output). `gcc -E` equivale al comando `cpp`.
- `gcc -S esempio.c`

  invoca precompilatore e compilatore e salva il file assembly in `esempio.s`
- `gcc -c esempio.c`

  invoca precompilatore, compilatore e assembler e salva il file in linguaggio macchina in `esempio.o`, può agire anche sul file `esempio.s`, invocando solo l'assembler.

Uso di gcc (compilatore GNU)

Comando	Azione	Files accettati in input	File di output

`g++`	Produce l'eseguibile	`.cc (sorgente)`
		`.s (assembly)`
		`.o (linguaggio macchina)`
			`a.out`, modificabile con l'opzione `–o`

`g++ -c`	Produce il file in linguaggio macchina	`.cc (sorgente)` `.s (assembly)`	`.o (linguaggio macchina)`
`g++ -s`	Produce il file in assembly	`.cc (sorgente)`	`.s (assembly)`
`g++ -E`	Applica il precompilatore	`.cc (sorgente)`	Output su schermo

esecuzione

- una volta compilato con successo un programma diventa eseguibile
- per lanciarne l'esecuzione e' sufficiente dal terminale Linux digitare il nome dell'eseguibile (`a.out` o `esempio`)
- nota: normalmente la directory corrente e' inserita in una variabile di sistema (PATH) che contiene tutte le directory in cui possono trovarsi dei comandi da eseguire (ad esempio /bin). Quando la directory corrente non e' nel PATH il sistema non trova il vostro eseguibile e vi risponde `a.out: command not found`
- se vi succede chiamatemi e modifichero' il vostro PATH
- in generale potete comunque sempre forzare il sistema ad eseguire un qualsiasi comando indicandogliene l'indirizzo completo o quello relativo:

```
./a.out
```