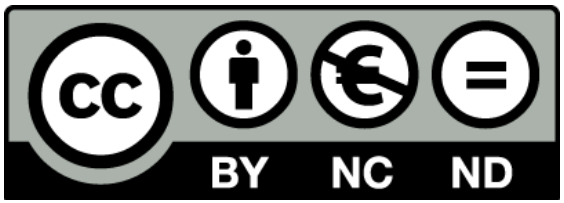
Laboratorio di Programmazione di Sistema

Programmazione Procedurale 1

Luca Forlizzi, Ph.D.

Versione 23.1



Luca Forlizzi, 2023

© 2023 by Luca Forlizzi. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode.

La Programmazione Procedurale

- Nella scrittura di un programma, sorge spesso l'esigenza di svolgere uno stesso compito in parti diverse del programma
- Ciò si può fare, in modo banale, replicando in più punti del programma la sequenza di istruzioni che svolge tale compito
- Tale approccio ha diversi aspetti critici
  - scrivere più volte le stesse istruzioni nel codice sorgente è tedioso
  - aumenta il rischio di commettere errori nella scrittura
  - il codice eseguibile del programma contiene più repliche della stessa sequenza di istruzioni, ciascuna delle quali occupa memoria: quindi il programma utilizza male la memoria
- Sin dall'inizio della storia della programmazione fu evidente che era opportuno un approccio migliore, che consentisse di scrivere una sola volta una sequenza di istruzioni S che necessita di essere eseguita in diverse parti del programma

La Programmazione Procedurale

- L'idea, introdotta già nei primi *HLL* e usata ancora oggi, è:
  1. Scrivere una sola copia della sequenza di istruzioni S, estrapolata dal resto della sequenza di istruzioni del programma
  2. Inserire, in ciascuno dei punti del programma in cui si vuole eseguire S, un comando "segnaposto", detto *costrutto di chiamata*, il cui effetto è che la abstract machine inizi ad eseguire S
  3. Inserire, in ciascun punto terminale di S, un comando, detto *costrutto di ritorno*, il cui effetto è che la abstract machine, dopo aver terminato l'esecuzione di S, continui l'esecuzione del programma con l'istruzione che segue il costrutto di chiamata che ha determinato la attuale esecuzione di S
- L'utilizzo di questa idea ha un significativo impatto sul modo in cui si progettano e scrivono programmi: per questo motivo viene considerato l'origine di uno stile fondamentale, detto *Programmazione Procedurale*

Procedura

- La sequenza di istruzioni S viene detta, a seconda del linguaggio impiegato, *metodo*, *procedura*, *funzione*, *routine*, *sottoprogramma* o *subroutine* (in particolare i due ultimi termini sottolineano il fatto che S è una parte del programma principale relativamente indipendente dal resto)
  - Il linguaggio Java e la maggior parte dei linguaggi orientati agli oggetti, usano il termine metodo
  - Il linguaggio C usa il termine funzione
  - Negli *ASM* sono prevalenti routine e subroutine
- In LPS, usiamo il termine *procedura* nelle affermazioni di carattere generale, indipendenti dal linguaggio; invece nel contesto linguaggi *ASM* usiamo il termine *routine* e per Java e C usiamo i rispettivi termini tecnici

Chiamata di una Procedura

- L'avvio dell'esecuzione di una procedura, viene detto *chiamata* o *invocazione* della procedura
- I costrutti di chiamata prendono diverse forme, in base al linguaggio considerato
- Nei linguaggi imperativi, i casi più comuni sono
  - *istruzioni di chiamata*, presenti, ed esempio, negli *ASM*, in Pascal, Fortran
  - *operatori di chiamata*, presenti, ad esempio in C, Java, Pascal, Fortran
- Quando non vi sono ambiguità, un costrutto di chiamata può essere detto, semplicemente, *chiamata*
- In relazione ad un costrutto di chiamata C_{CALL}
  - la procedura che contiene C_{CALL} viene detta *chiamante*
  - la procedura che viene chiamata da C_{CALL} viene detta *chiamata*

Ritorno da una Procedura

- Il termine dell'esecuzione di una procedura e il conseguente inizio dell'esecuzione del costrutto che segue il costrutto di chiamata, viene detto *ritorno* dalla procedura
- Nei linguaggi imperativi, un costrutto che indica alla abstract machine di effettuare il ritorno da una procedura, è tipicamente un'istruzione, detta *di ritorno*
- Quando non vi sono ambiguità, un'istruzione di ritorno può essere detta, semplicemente, *ritorno*

Attivazione di una Procedura

- Nel corso di un'esecuzione del programma, uno stesso costrutto di chiamata può essere eseguito più volte
- La conseguenza di ogni distinta esecuzione di un costrutto di chiamata, è che la abstract machine prosegue l'esecuzione del programma dalla prima istruzione della procedura chiamata
- Ciascuno di tali eventi è distinto dagli altri, e viene detto *esecuzione della procedura* o *attivazione della procedura*

Attivazione di una Procedura

- In relazione a ciascuna distinta esecuzione di un costrutto di chiamata C_{CALL}
  - l'attivazione che esegue C_{CALL} viene detta *attivazione chiamante*
  - l'attivazione che inizia a seguito dell'esecuzione di C_{CALL}, viene detta *attivazione chiamata*
- Quando viene eseguito un costrutto di chiamata C_{CALL}
  - l'attivazione chiamante viene sospesa e la abstract machine inizia ad eseguire le istruzioni dell'attivazione chiamata
  - quando l'attivazione chiamata termina, la abstract machine riprende l'esecuzione dell'attivazione chiamante, a partire dal costrutto che segue C_{CALL}

Il Programma nella Programmazione Procedurale

- Nello stile della Programmazione Procedurale, un programma non viene più concepito come un'indistinta sequenza di istruzioni, ma come un insieme di procedure
- L'esecuzione di un programma inizia con un'attivazione di una specifica procedura, detta *procedura principale*, identificata in qualche modo dalle regole del linguaggio
- La procedura principale può chiamare altre procedure, e ogni attivazione di procedura può, a sua volta, effettuare altre chiamate

Il Programma nella Programmazione Procedurale

- Nel corso dell'esecuzione di un programma, ogni procedura può essere attivata più volte
- Anche la procedura principale può essere attivata più volte, in quanto anche tale procedura può essere chiamata, da altre procedure o da se stessa
- La prima attivazione, in ordine di tempo, della procedura principale, viene detta *attivazione principale*
- L'attivazione principale ha inizio prima di ciascuna altra attivazione e termina dopo ciascuna altra attivazione
- Il termine dell'attivazione principale coincide con la fine dell'esecuzione dell'intero programma

Il Programma nella Programmazione Procedurale

- Un *albero di chiamata* (in inglese *call tree*) è un diagramma ad albero che rappresenta un'esecuzione di un programma
  - Il diagramma contiene un punto per ciascuna attivazione di procedura che si verifica durante l'esecuzione del programma
  - Per ogni attivazione di procedura, si traccia una freccia diretta dall'attivazione chiamante all'attivazione chiamata
  - Si ordinano le frecce uscenti da ciascuna attivazione, in base all'ordine in cui si sono verificate
- Una visita dell'albero di chiamata, che rispetti l'ordine delle frecce uscenti da ciascun punto, permette di ricostruire la sequenza di attivazioni eseguite dal programma
- L'attivazione principale è la radice dell'albero di chiamata

Il Programma nella Programmazione Procedurale

- Nella visita
  - ogni chiamata di procedura corrisponde a spostarsi da un'attivazione all'altra seguendo il verso di una freccia
  - ogni ritorno da una procedura corrisponde a spostarsi da un'attivazione all'altra andando nel verso opposto a quello della freccia
- La terminazione del programma corrisponde a terminare la visita ritornando all'attivazione principale, dopo aver seguito tutte le frecce uscenti da tale attivazione

Procedura Foglia

- Una procedura che non contiene costrutti di chiamata, viene detta *procedura foglia*
- Una procedura che, al contrario, può eseguire costrutti di chiamata, viene detta *procedura non-foglia*
- In ciascun albero di chiamata di un programma, tutte le attivazioni di procedure foglia non hanno frecce uscenti da esse
- Tuttavia, in alcuni alberi di chiamata, vi possono essere attivazioni di procedure non-foglia che non hanno frecce uscenti da esse

Procedura Foglia

- In Code1, `f2` è una funzione foglia, mentre `f1` è non-foglia

Code1

```
#include <stdio.h>
int x;
void f2( void ) { x = -x; }
void f1( void ) {
   scanf( "%d", &x );
   if ( x < 0 ) f2( ); /* f1 chiama f2: f1 e' non-foglia */
   x *= 2;
   return;
}
```

- Un'attivazione di `f1` potrebbe non chiamare `f2`
- Dunque, in alcuni alberi di chiamata, le attivazioni di `f1` potrebbero non avere frecce da esse uscenti

Vantaggi della Programmazione Procedurale

- Si evita di scrivere più volte, nel codice sorgente, una sequenza di istruzioni
- Si evita di replicare più volte, nel codice eseguibile, una sequenza di istruzioni *LM*
- Le procedure diventano un mezzo per suddividere un programma in parti che rispecchiano la logica usata per progettare il programma
- Negli *HLL*, si può dare a ciascuna procedura un nome che suggerisce quale sia il compito da essa effettuato
- Il programma, suddiviso in parti relativamente indipendenti dal punto di vista logico, diventa più leggibile
- Le procedure possono essere riutilizzate in altri programmi

Basi di Programmazione Procedurale in C

- In C, una funzione attribuisce un *nome* ad un insieme di dichiarazioni di variabile e di istruzioni che formano una sequenza
- Se **F** è una funzione, l'associato insieme di dichiarazioni e istruzioni viene chiamato *corpo della funzione* **F**
- Le funzioni in C sono simili ai metodi in Java
- La principale differenza è che i metodi Java appartengono ad oggetti o a classi, mentre le funzioni C sono entità indipendenti da altre
- Altre differenze saranno evidenti nel seguito di questa presentazione e in future presentazioni

Basi di Programmazione Procedurale in C

- Le funzioni in C possono comunicare tra loro attraverso vari meccanismi
- I principali sono il *passaggio di parametri* e la *restituzione del risultato*, analoghi agli omonimi meccanismi dei metodi Java
- In LPS, chiamiamo *funzione semplice*, una funzione che gode delle seguenti proprietà
  - è una funzione foglia
  - non riceve parametri
  - non restituisce un risultato
- Questa presentazione si focalizza sulle funzioni semplici, che sono le più facili da tradurre in *ASM*

Basi di Programmazione Procedurale in C

- Una *definizione di funzione* è un costrutto che stabilisce l'esistenza di una funzione in un programma
- Essa viene scritta al di fuori di ogni altro costrutto e indica il nome e il corpo di una funzione
- In una versione *hosted* di C Standard, la funzione principale, in ogni programma C, ha nome `main`

Basi di Programmazione Procedurale in C

- La sintassi della definizione di una funzione che non riceve parametri e non restituisce un risultato, è descritta nello Schema1, in cui **Name** è il nome della funzione e **Body** è un insieme di costrutti C disposti in sequenza
- Si noti, in Schema1, la presenza di due istanze della keyword `void`
  - la prima indica che la funzione non restituisce risultato
  - la seconda indica che la funzione non ha parametri

Schema1

```
void Name ( void )
{
  Body
}
```

Basi di Programmazione Procedurale in C

- Per effettuare una chiamata di funzione si scrive il nome della funzione, seguito dall'*operatore di chiamata di funzione*, ovvero la coppia di parentesi `( )`, tra cui, come mostreremo in future presentazioni, possono essere inseriti gli argomenti della chiamata
- Il ritorno dalla funzione viene effettuato quando, nel corso dell'esecuzione della funzione, si verifica uno dei seguenti eventi
  - si raggiunge la fine del corpo della funzione
  - si esegue un'istruzione `return` contenuta nel corpo della funzione

Basi di Programmazione Procedurale in C

- Code2 mostra un programma C che definisce e chiama una funzione semplice

Code2

```
int a = 4, c = -2;
void change_values( void ) {
 a += 2;
 c += 5;
 return;
}

int main( void ) {
 /* altre istruzioni */
 change_values();
 /* altre istruzioni */
 change_values();
 /* altre istruzioni */
}
```

Basi di Programmazione Procedurale in *ASM*

- Nei linguaggi *ASM*, una routine è una sequenza di istruzioni
- La chiamata di una routine consiste in un salto alla prima istruzione che forma la routine
- Il ritorno della routine avviene tramite un salto all'istruzione immediatamente successiva, in ordine testuale, all'istruzione di salto che ha effettuato la chiamata
- Poiché in un programma ci possono essere diverse istruzioni di salto che effettuano la chiamata della stessa routine, è necessario un meccanismo per individuare in modo dinamico, ad ogni esecuzione della routine, qual'è l'istruzione immediatamente successiva a quella che ha chiamato l'esecuzione corrente della routine

Basi di Programmazione Procedurale in *ASM*

- Il meccanismo che consente di individuare l'istruzione immediatamente successiva a quella che ha effettuato la chiamata della routine, utilizza l'indirizzo di tale istruzione, detto *indirizzo di ritorno* della chiamata
  - Per ogni chiamata di routine, si memorizza, in una parola di lunghezza adeguata, l'indirizzo di ritorno
  - Al termine dell'esecuzione della routine, si legge da tale parola l'indirizzo di ritorno e si salta ad esso, mediante un'istruzione di salto con destinazione dinamica

Basi di Programmazione Procedurale in *ASM*

- La maggior parte degli *ASM* dispone di speciali istruzioni di salto, dette *istruzioni di chiamata di routine*, che rendono efficiente la chiamata di una routine
- Un'istruzione di chiamata di routine J_{CALL}, oltre ad effettuare il salto, memorizza automaticamente in una parola l'*indirizzo di ritorno della chiamata*, ovvero l'indirizzo dell'istruzione successiva a J_{CALL} in ordine testuale
- Tale indirizzo viene letto e usato come indirizzo destinazione, dall'istruzione di salto J_{RET} che effettua il ritorno dalla routine
- Molti *ASM*, dispongono di istruzioni di salto speciali anche per eseguire il ritorno dalla routine

Routine in MIPS32

- Tranne una, le istruzioni di chiamata di routine di MIPS32 memorizzano l'indirizzo di ritorno dalla routine nel registro 31
- Proprio per questo motivo, MIPS32 attribuisce al registro 31 il nome simbolico `ra`, da "Return Address"
- Dunque `ra` è un registro general purpose with special function
- MIPS32 usa per le istruzioni di chiamata nomi simbolici che abbreviano le espressioni inglesi "jump and link" o "branch and link" perché tali istruzioni, oltre a saltare, stabiliscono un legame tra la routine chiamata e l'istruzione da eseguire dopo il termine della routine

Routine in MIPS32

- Le più comuni istruzioni di chiamata di routine di MIPS32-MARS, sono
  - `bgezal`
  - `bltzal`
  - `jal`
  - `jalr`
- L'indirizzo di ritorno dalla routine viene memorizzato
  - nel registro `ra` nel caso delle prime 3 istruzioni
  - in un registro specificato come operando nel caso di `jalr`

Routine in MIPS32

- Per effettuare il ritorno dalla routine è possibile usare l'istruzione `jr` che, come sappiamo, è una istruzione di salto incondizionato con destinazione dinamica che ha come operando un registro, del quale `jr` usa il contenuto come indirizzo di destinazione del salto
- Di conseguenza MIPS32 non ha istruzioni speciali per eseguire il ritorno da una routine, diversamente da altri *ASM*, come MC68000