

Programmazione

**Amato Flora**

a.a. 2023-2024

**Author**

Alessio Romano

August 20, 2024

## Contents

<b>1</b>	<b>Sottoprogrammi</b>	<b>3</b>
1.1	Funzionamento di un sottoprogramma . . . . .	3
1.2	Tipi di sottoprogrammi e astrazione sul controllo . . . . .	4
1.3	Interfaccia . . . . .	4
1.4	Modi di scambio dei paramentri . . . . .	5
<b>2</b>	<b>Compilazione separata</b>	<b>5</b>
2.1	Inclusione di file . . . . .	6
<b>3</b>	<b>Operazioni su File</b>	<b>7</b>
3.1	Stream . . . . .	7
3.1.1	Tipi di stream . . . . .	7
3.1.2	Connessione di uno stream . . . . .	7
3.1.3	Stream standard di I/O . . . . .	8
3.2	File . . . . .	8
3.2.1	Posizionamento in lettura e scrittura . . . . .	8

# 1 Sottoprogrammi

I **sottoprogrammi** costituiscono un meccanismo di astrazione procedurale (astrazione sul controllo).

Si tratta di sezione di codice che non vengono eseguite autonomamente una volta implementate, ma su chiamata di un programma esterno. Essi svolgono funzioni di utilità generale (ex. librerie), o semplicemente aiutano a sviluppare codice ben strutturato e modulare. Utilizzare sottoprogrammi comporta vari vantaggi

- Suddivisione del programma in unità significative
- Testo di ogni unità più breve
- Riutilizzo del codice
- Migliore leggibilità
- Sviluppo top-down del software

## 1.1 Funzionamento di un sottoprogramma

Ai sottoprogrammi vengono assegnate determinate responsabilità, che includono la risoluzione di un sottoproblema. I sottoprogrammi hanno due caratteristiche principali

- Un **nome** per effettuare la chiamata da qualunque punto del programma chiamante
- Un unico **entry-point** ed un unico **exit-point** (secondo i principi della programmazione strutturata).

Il flusso di controllo è ceduto dal modulo chiamante al sottoprogramma da una istruzione di **chiamata**. L'interazione tra il sottoprogramma e il programma

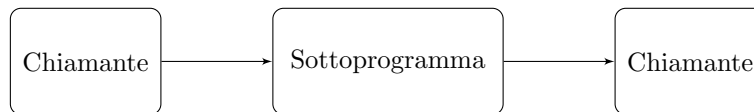


Figure 1: chiamata a sottoprogramma

chiamante avviene mediante uno scambio di dati. I dati possono essere di ingresso, di uscita o di ingresso/uscita.

Lo scambio avviene mediante paramentri, ne esistono di due tipi:

- **Paramentri formali:** nomi delle variabili passate alla funzione
- **Parametri attuali:** valori delle variabili passate alla funzione

I parametri formali e attuali devono coincidere in numero, posizione e tipo.

```
int b;  
stampa(b);  
// b parametro effettivo  
  
void stampa (const int x){  
    // x parametro formale  
    cout << x;  
}
```

Quando si esegue una chiamata ad un sottoprogramma, per la **regola della sostituzione**, l'effetto che si ottiene é l'esecuzione del sottoprogramma con la sostituzione di ciascun parametro effettivo a ciascun parametro formale

## 1.2 Tipi di sottoprogrammi e astrazione sul controllo

Definiamo 2 diversi tipi di sottoprogrammi

- **Funzioni:** esprimono l'astrazione matematica di una funzione, calcolano un risultato valutato a partire dai valori che vengono forniti. Una funzione ritorna un unico valore di tipo atomico attraverso l'istruzione *return*
- **Procedure:** esprimono l'astrazione di un insieme di azioni che producono degli "effetti" ma non "ritornano" alcun risultato

I sottoprogrammi in genere, implementano un **astrazione sul controllo**, ossia possono essere considerate **black box**.

Essi svolgono un compito, offrendo un servizio a chi li utilizza, ma chi chiama il sottoprogramma non ha visibilit  dell'implementazione di quest'ultimo.

L'interazione tra sottoprogramma ed ambiente esterno avviene mediante una precisa interfaccia costituita dall'intestazione.

## 1.3 Interfaccia

L'interfaccia di un sottoprogramma deve specificare

- Il nome del sottoprogramma (deve essere significativo della funzionalit  di quest'ultimo)
- I parametri di input
- I parametri di output

```
int incr(int); // intestazione  
  
incr(x){  
    x++  
    return x  
}
```

## 1.4 Modi di scambio dei parametri

Esistono diversi modi per effettuare lo scambio tra parametri effettivi e parametri globali.

- **passaggio per valore:** viene realizzato dal compilatore copiando il valore del parametro effettivo nel corrispondente parametro formale. La modifica di un parametro passato per valore, non verrà vista dal programma chiamante.
- **passaggio per indirizzo:** viene realizzato dal programmatore, e consiste nel passare per valore l'indirizzo della variabile. La modifica avviene dunque sulla variabile stessa e dunque sarà visibile anche dal programma chiamante
- **passaggio per riferimento:** un parametro riferimento è un alias della variabile argomento, ha lo stesso funzionamento di un passaggio per indirizzo ma è più efficiente in quanto evita la copia del valore
  - Il parametro formale è un puntatore che contiene l'indirizzo del parametro effettivo
  - Il passaggio dell'indirizzo è a cura del compilatore

```
void incr(int &v){
    v+= 1
}

int a = 4
incr(a)
// a = 5
```

Possono essere passati per valore solo i parametri di ingresso. I parametri in uscita e quelli di ingresso/uscita, devono essere passati per riferimento

	valore	indirizzo	riferimento
input	sì	possibile	possibile
output	no	sì	sì
input/output	no	sì	sì

In c e c++ di default gli array sono scambiati per indirizzo, le variabili atomiche per valore

## 2 Compilazione separata

È buona norma tenere separata la specifica di un modulo dalla sua implementazione.

- Un programma utente di un modulo A, deve conoscerne la specifica, ma non l'implementazione.
- Ciò avviene attraverso l'utilizzo di un file di intestazione (\*.h) contenente le dichiarazioni che costituiscono l'interfaccia A, ed un file separato per l'implementazione di A.
- Siccome ogni modulo deve essere autoconsistente, ovvero contenere tutte le informazioni necessarie per la compilazione, l'header deve essere incluso nella implementazione di ogni modulo utente (mediante direttiva include in c, c++)

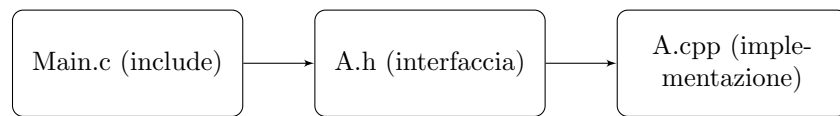


Figure 2: Utilizzo di un modulo

Un programma in C++ consiste in più file sorgente compilati separatamente in file oggetto, successivamente collegati da un "linker", che produce la forma eseguibile del programma

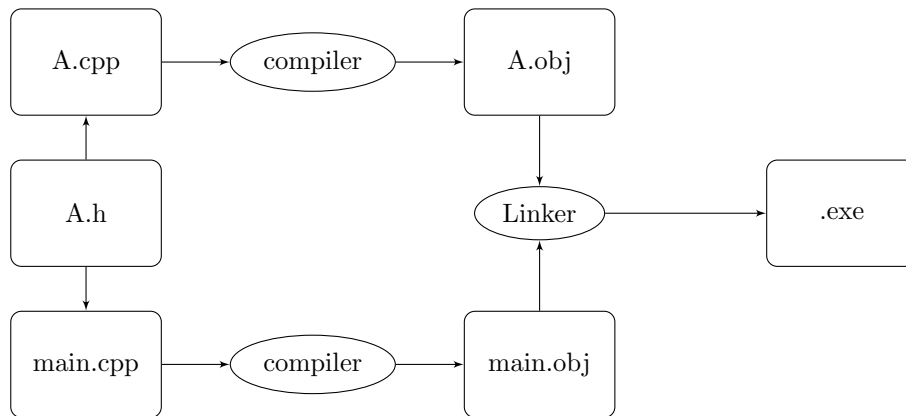


Figure 3: Compilazione separata

## 2.1 Inclusione di file

In C, il preprocessore tramite la direttiva **include**, può ricercare il file indicato in alcune directory standard o definite al momento della compilazione ed espanderlo testualmente in sostituzione della direttiva. In C la direttiva include si applica in due diverse forme

```
#include <nome-file >
#include "nome-file"
```

- Nel primo caso, il nomefile viene ricercato nelle directory standard definite dall'implementazione ed in altre che sono specificate al momento della compilazione
- Nel secondo caso, il nomefile viene ricercato nella directory corrente, e successivamente se non viene trovato, nelle directory standard e quelle specificate al momento della compilazione come nel primo caso

## 3 Operazioni su File

### 3.1 Stream

Il sistema di I/O di C e C++ si basa sul concetto di stream. Per stream, si intende un flusso di dati (sorgente o destinazione) che può essere associato ad una periferica o alla stessa maniera ad un file proveniente dalla memoria di massa.

In altre parole uno stream è un interfaccia comune a diversi dispositivi di I/O che permette di:

- rendere la scrittura dei programmi indipendenti dal particolare dispositivo impiegato
- Semplificare i problemi di portabilità dei programmi.

Lo stream consente di definire un'interfaccia semplice di astrazione di un dispositivo generico e uniforme verso l'utente

#### 3.1.1 Tipi di stream

La libreria dedicata alle operazioni su gli stream, sia in c che in c++ supporta stream di testo e stream binari.

- Uno stream di **testo** è una sequenza di linee, ciascuna formata da caratteri e chiusa dal carattere "\ n" Uno stream **binario** è una sequenza di byte, che registrano dati interni (di qualsiasi tipo) chiusa da un terminatore **end-of-file** garantendo che esista corrispondenza tra quello che viene scritto e successivamente letto sullo stesso sistema

#### 3.1.2 Connessione di uno stream

Nel caso di operazioni di I/O verso un **dispositivo**, lo stream viene connesso tramite una operazione di apertura,ossia viene creato un oggetto e linkato ad uno stream. La connessione viene interrotta con un'operazione di chiusura, e possiamo definire due operazioni sullo stream:

- **Estrazione da un flusso:** la ricezione di dati da un dispositivo di input
- **Inserimento in un flusso:** la trasmissione di dati ad un dispositivo di uscita

Nel caso di operazioni di I/O verso la **memoria di massa**, sia ha una generalizzazione delle operazioni primarie. Ad esempio in C e C++ la libreria I/O mette a disposizione tre classi

- **ifstream:** operazioni di input
- **ofstream:** operazioni di output
- **fstream:** operazioni di aggiornamento

### 3.1.3 Stream standard di I/O

In C++ un programma comunica con l'esterno mediante i seguenti stream standard

- **stdin:** standard input (associato di default alla tastiera)
- **stdout:** standard output (associato di default al video)
- **stderr:** standard output per i messaggi (associato al video)

Gli stream sovraelencati sono rappresentamente il canale di immissione dati, uscita dei risultati, riporto delle eventuali situazioni di errore. Questi stream sono collegati alle variabili globali `cin`, `cout`, `cerr`, e si possono utilizzare attraverso gli **operatori di flusso** `>>` (estrazione) e `<<` (inserimento).

## 3.2 File

Per memorizzare un dato su un'unità di memoria di massa viene utilizzata una variabile di tipo stream chiamata **File**. Esistono due varianti

- **File di testo:** Una sequenza di caratteri che durante il trasferimento può subire conversioni a seconda delle necessità e dell'ambiente di destinazione
- **File binari:** una sequenza di Byte

### 3.2.1 Posizionamento in lettura e scrittura

La libreria `fstream` gestisce due puntatori, uno per la lettura e uno per la scrittura che possono essere riposizionati usando le funzioni apposite

- Posizionamento e interrogazione del puntatore in lettura:

```
seekg() //
tellg() //
```

- Posizionamento e interrogazione del puntatore in scrittura:

```
seekp ( )  //  
tellp ( )  //
```