I principi guida della buona programmazione

Programmazione in grande

Lo sviluppo di sistemi software complessi (programmazione in grande o “in the large”) richiede di:

1. Suddividere il lavoro tra diversi gruppi di lavoro e di contenere i tempi e i costi di sviluppo che sono in generale molto alti.
2. Mantenere alta la qualità del software in termini di:

   affidabilità;

   prestazioni;

   modificabilità;

   estensibilità;

   riutilizzo.

La **programmazione modulare** è un necessario approccio alla programmazione in grande, in quanto consente gestire la complessità del sistema da realizzare suddividendolo in parti tra loro correlate.

Modularizzazione

La modularizzazione è la suddivisione in parti (moduli) di un sistema, in modo che esso risulti più semplice da comprendere e manipolare.

Essa determina la definizione di una “architettura”, che descrive l’organizzazione del sistema in parti e le interconnessioni tra di esse.

Gran parte dei sistemi complessi sono modulari. Ad esempio, un elaboratore composto da diversi sotto-sistemi:

- sottosistema di elaborazione;
- sottosistema di memorizzazione;
- sottosistema di comunicazione;
- rete di interconnessione.

Modularizzazione e Astrazione

La suddivisione di un sistema in parti richiede che ciascuna di esse realizzi un aspetto o un comportamento ben definito all'interno del sistema.

E' necessario a questo scopo che venga effettuato un processo di astrazione

Il concetto di modulo

Un modulo in un sistema software è un componente che può essere utilizzato per realizzare una astrazione.

In generale può esportare all'esterno:

- servizi/funzioni ("astrazione sul controllo");
- dati ("astrazione sui dati");
- identificativi.

A sua volta può importare dati, funzionalità e identificativi da altri moduli.

Definisce un AMBIENTE DI VISIBILITA'.

Realizzazione di un modulo

Un modulo è dotato di una chiara separazione tra:

- interfaccia;
- corpo.

L’interfaccia specifica “cosa” fa il modulo e “come” si utilizza. Deve essere visibile all’esterno del modulo per poter essere utilizzata dall’utente del modulo al fine di usufruire dei servizi/dati esportati dal modulo.

Il corpo descrive “come” l’astrazione è realizzata e contiene l’implementazione delle funzionalità/strutture dati esportate dal modulo che sono nascoste e protette all’interno del modulo. L’utente può accedere ad esse solo attraverso l’interfaccia.

Modularizzazione in C++

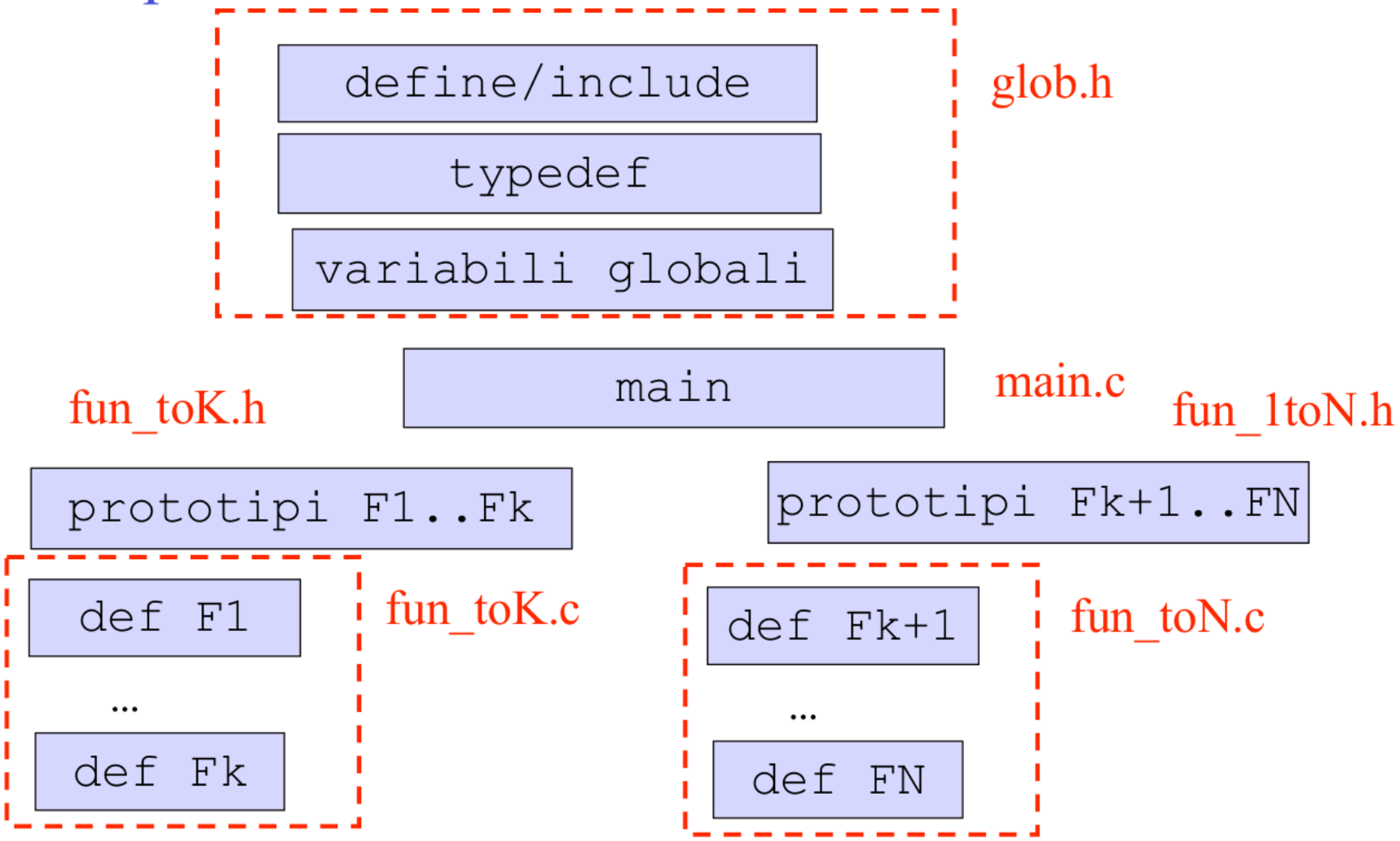
L’uso disciplinato di alcuni meccanismi del linguaggio C++ consente una corretta strutturazione di un programma in moduli.

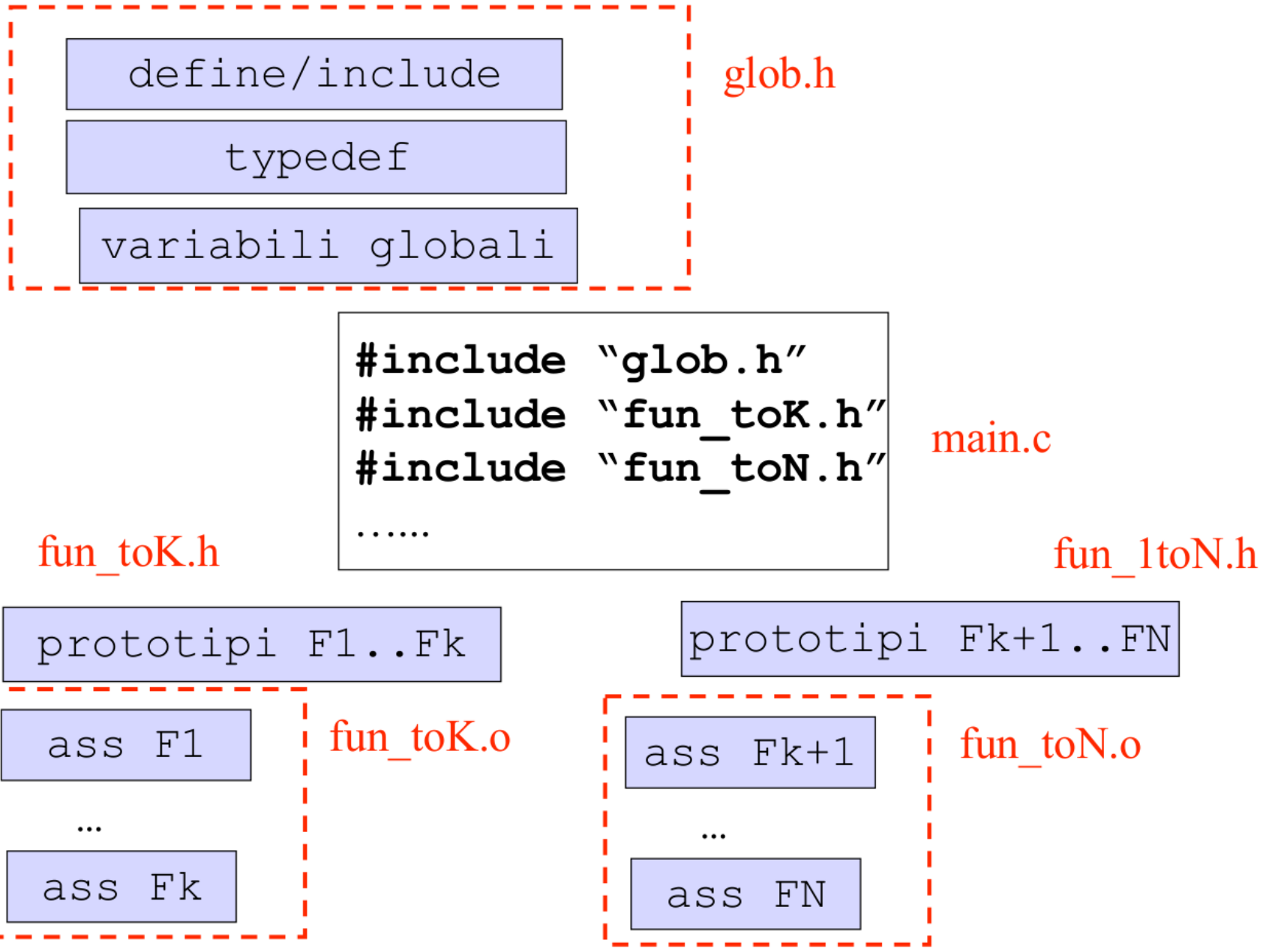
Tra i principali meccanismi vi sono:

- la compilazione separata,
- l’inclusione testuale,
- l’uso dei prototipi di funzioni,
- l'uso dei namespace.

Compilazione separata (3)

- Tipicamente :




define/include
typedef
variabili globali
glob.h
#include "glob.h"
#include "fun_toK.h"
#include "fun_toN.h"
.....
main.c
fun_toK.h
fun_1toN.h
prototipi F1..Fk
prototipi Fk+1..FN
ass F1
...
ass Fk
fun_toK.o
ass Fk+1
...
ass FN
fun_toN.o

Information hiding

Consiste nel nascondere e proteggere (incapsulare) alcune informazioni di una entità all'interno del modulo.

Alle informazioni protette l'accesso da parte degli utilizzatori del modulo è fornito in maniera controllata.

L'accesso ad esse è possibile solo attraverso l'interfaccia

Information hiding: criteri

Valutare con attenzione cosa esportare e cosa nascondere.

Le interfacce devono essere:

- minimali, cioè devono esporre solo ciò che è strettamente necessario;
- stabili, cioè possibilmente non devono subire cambiamenti nel tempo: se l’interfaccia non cambia, le informazioni nascoste possono essere modificate senza che questo influisca sulle altre parti del sistema di cui l’entità fa parte

Come realizzeremo l'information hiding

- Avete “intuito” a IP che un buon strumento linguistico per realizzare i principi di encapsulation e information hiding sono le classi.
- Tuttavia, nel corso di ASD non utilizzeremo le classi (a parte alcune utili classi predefinite, quali i vector) e ci limiteremo ad utilizzare principalmente il frammento imperativo del C++

Nota implementativa

- Per offrire comunque un buon meccanismo di incapsulamento senza usare le classi, sfrutteremo i namespace.

Usare i namespace in C++

I namespace (in italiano spazio dei nomi) sono stati introdotti nel linguaggio C++ per evitare collisioni fra nomi. Inizialmente tutti i nomi di funzioni, variabili, classi venivano inclusi in uno spazio di nomi globale. Quando vi erano due o più elementi con lo stesso nome allora si incorreva in una collisione. Con l'introduzione dei namespace è possibile specificare a quale namespace appartiene una particolare funzione, variabile o classe.

Per indicare al compilatore che ad esempio l'identificatore cout deve essere cercato nello spazio dei nomi standard (std) anteponiamo a cout, std:: , dove:

std è lo spazio dei nomi standard

:: è l'operatore di risoluzione dell'ambito

Usare i namespace in C++

Anteponendo std:: a cout diciamo che stiamo qualificando esplicitamente cout.

Se non vogliamo qualificare esplicitamente ogni identificatore del namespace, possiamo utilizzare in alternativa la seguente espressione dopo l'inclusione della libreria iostream:

using namespace std;

Dichiarare i namespace in C++

- I nomi raggruppati dal namespace hanno visibilità locale al namespace, in questo senso sono “incapsulati” nel namespace e non sono visibili all’esterno; in questo modo si evitano i conflitti nel caso gli stessi nomi si ripetano nell’ambito del programma.

- Il nome del namespace (che a sua volta è un identificativo) è invece un nome globale e può essere usato al di fuori del namespace.

Per vedere i namespace all'opera, si veda il codice realizzato in classe

Numeri (pseudo-)casuali

Generazione di un numero casuale tra inizio e fine compresi

```
int numCasuale = inizio+rand()%(fine-inizio+1);
```

Es: inizio = 3; fine = 7

numCasuale = 3 + rand()%(7-3+1)