Programmazione orientata agli oggetti
Introduzione

Un nuovo modello

- Il corso di fondamenti di informatica L-B ha come oggetto un nuovo modello di programmazione (paradigma): la programmazione orientata agli oggetti o OOP (Object-Oriented Programming)
- La programmazione procedurale, il modello su cui si basa il C, è adatta per applicazioni di dimensioni medio-piccole ma entra in crisi quando si superano le 100.000 istruzioni
- La programmazione orientata agli oggetti è invece un ottimo strumento per gestire la complessità e consente di realizzare applicazioni costituite da milioni di istruzioni

La crescita della complessità

- Il modello procedurale è nato alla fine degli anni 60 ed era pensato per le esigenze dell'epoca
- Il programma di gestione di Saturno V, il razzo che ha portato l'uomo sulla luna nel 1969, era composto da 16.000 istruzioni
- Il programma di gestione di un moderno aereo è composto da due milioni di istruzioni
- E' evidente che le metodologie e gli strumenti devono adeguarsi a questo salto di complessità

OOP: gli strumenti

- Per lavorare con questo nuovo modello impareremo a usare due nuovi strumenti:
- Un linguaggio di programmazione: Java, uno tra i più diffusi linguaggi object oriented
  - Java è un linguaggio moderno creato nel 1995 e si presta molto bene all'apprendimento dell'OOP
- Un linguaggio di modellazione: UML che ci consentirà di rappresentare graficamente l'organizzazione dei programmi che scriveremo
  - UML è uno standard, accettato praticamente da tutti, per rappresentare graficamente i concetti legati al modello OOP
  - UML è piuttosto complesso e ne useremo un sottoinsieme molto semplice

Limiti del modello procedurale

- Il modello procedurale è basato su un dualismo di fondo
- Un programma è composto da due tipi di entità: strutture dati e procedure
- Strutture dati: entità passive che contengono informazioni
- Procedure: entità attive che modificano le informazioni
- Il problema è che queste due entità sono scollegate fra di loro: se dichiariamo una variabile globale, tutte le procedure di un'applicazione possono modificarla senza controllo: manca il concetto di protezione dei dati
- In un'applicazione complessa il fatto che in ogni punto si possa modificare qualunque dato può facilmente generare una situazione incontrollabile

Modularizzazione e decomposizione

- Lavorare in questo modo è come costruire un computer mettendo tutti i componenti su una sola scheda e dare la possibilità di collegare un componente con tutti gli altri senza alcuna limitazione
- L'industria dell'hardware è riuscita a costruire computer sempre più complessi perché ha adottato un concetto di modularizzazione a più livelli:
  - I singoli componenti sono racchiusi in circuiti integrati
  - I circuiti integrati vengono montati su schede
  - Le schede vengono montate su rack per comporre il computer
- Un problema complesso come la costruzione di un calcolatore diventa gestibile perché il problema viene decomposto in problemi più semplici: costruzione di una scheda, di un circuito integrato…

Astrazione

- Una struttura organizzata a livelli consente anche di concentrarsi solo sugli aspetti importanti ignorando i dettagli che sono già stati affrontati e risolti ai livelli inferiori
- Quando progettiamo una scheda ci concentriamo solo sui meccanismi di collegamento tra i vari circuiti integrati e ignoriamo completamente i dettagli di progettazione dei circuiti integrati
- Quando progettiamo un computer ci concentriamo solo su come comporre fra di loro le schede e ignoriamo completamente i dettagli di progettazione delle schede
- Questo meccanismo prende il nome di astrazione ed è un potente strumento per gestire la complessità

Interfaccia e implementazione

- Un circuito integrato nasconde il suo contenuto ed espone solo un numero limitato di piedini
- Si può sostituire un circuito con un altro purché i segnali elettrici ai piedini rimangano gli stessi
- Si ha quindi una separazione fra quello che un componente è in grado di fare (interfaccia) e come lo fa (implementazione)
- L'implementazione è un dettaglio interno che non influisce sul comportamento del sistema
- Lo stesso succede tra computer e periferiche: noi possiamo collegare ad un computer una qualsiasi stampante perché è stata definita un'interfaccia standard e praticamente tutte le stampanti, indipendentemente dalla tecnologia costruttiva, si adeguano a questa interfaccia

Gestire la complessità: decomposizione e astrazione

- E' stato dimostrato che il numero massimo di elementi di informazione che un individuo può gestire simultaneamente è nell'ordine di 7 (+/- 2).
- Questo limite viene chiamato capacità di canale del meccanismo cognitivo umano.
- Attraverso la decomposizione suddividiamo un sistema in parti via via più piccole, che possono essere rifinite separatamente, ci troviamo a gestire contemporaneamente un numero limitato di informazioni e quindi possiamo superare il limite dato dalla capacità di canale
- Nell'incapacità di padroneggiare un oggetto complesso nella sua interezza, attraverso l'astrazione scegliamo di eliminare i suoi dettagli meno essenziali operando con un modello generalizzato e semplificato.
- Siamo sempre limitati dal numero di cose che siamo in grado di comprendere esattamente ma, grazie all'astrazione, usiamo elementi di informazione con contenuto via via crescente

Evoluzione del modello procedurale

- Anche lavorando in C, quindi con il modello procedurale ci si è presto resi conto che era necessario operare in modo modulare
- Nel tempo si sono quindi diffuse pratiche di progettazione e di programmazione basate sui concetti che abbiamo appena esposto
- Un bell'esempio di questo approccio è costituito dalla gestione dei file nella libreria standard del C
- Nella gestione dei file troviamo una serie di concetti estremamente interessanti

I file in C: esempio

```
/* Dichiariamo variabili di tipo puntatore a FILE
   (riferimenti) */
   FILE* f1,f2;
/* Con fopen apriamo i file: il sistema alloca risorse,
   "crea" una struttura per gestire il file e ci restituisce
   un puntatore (riferimento) */
   f1 = fopen("pippo.txt","r");
   f2 = fopen("pluto.txt","w");
/* Ottenuti i riferimenti li utilizziamo in tutte le
   operazioni successive */
   fscanf(f1,...);
   fprintf(f2,...);
/* Quando non ci servono più chiudiamo i file e il sistema
   libera le risorse allocate: i file "cessano di esistere" */
   fclose(f1);
   fclose(f2);
```


I file in C: astrazione

- Non abbiamo nessuna idea di come sia fatta una struttura dati di tipo FILE
- Abbiamo a disposizione una serie di funzioni (primitive) che ci permettono di operare sul file senza sapere come è fatto
- La struttura dati FILE potrebbe cambiare completamente e il nostro programma non ne risentirebbe minimamente
- C'è una netta separazione fra interfaccia e implementazione
- Non ci occupiamo minimamente dei dettagli implementativi ma ci concentriamo su un'astrazione

I file in C: incapsulamento

- Non possiamo agire direttamente sulla struttura dati
- Abbiamo a disposizione solo un “riferimento” alla struttura e possiamo operare su di esso solo attraverso le funzioni primitive
- Questo garantisce che, se le primitive sono state realizzate correttamente, il suo stato è sempre consistente
- Quindi la struttura dati è protetta
- Si dice che la struttura dati è ”incapsulata” e il meccanismo di protezione viene chiamato incapsulamento.

I file in C: dinamicità

- I file vengono gestiti in modo dinamico
- Non hanno un tempo di vita limitato e “automatico” come le variabili locali
- Non esistono per tutta la durata del programma come le variabili globali
- Vengono invece creati con fopen e distrutti con fclose
- Il loro tempo di vita viene quindi gestito da chi li utilizza
- All’atto della “creazione” viene allocata un’area di memoria che viene liberata nel momento della “distruzione”
- Come tutto il resto, anche l’allocazione/deallocazione memoria viene gestita dalle primitive e questo garantisce la consistenza

I file in C: istanze, stato e comportamento

- Grazie al meccanismo di creazione/distruzione è possibile lavorare in contemporanea con più file
- E' sufficiente dichiarare più variabili di tipo riferimento a FILE e invocare fopen() più volte memorizzando il valore di ritorno nelle diverse variabili
- Tutte le altre primitive prevedono come primo parametro un riferimento a FILE, e quindi è possibile operare indipendentemente sui vari file aperti
- Quindi: abbiamo diverse entità con un comportamento comune (determinato dalle primitive) ma con uno stato diverso (ogni variabile lavora su un file diverso, posizionato su una riga diversa ecc.)
- Abbiamo più istanze e l'insieme costituito dal tipo FILE e dalla funzione di creazione fopen() costituisce una "matrice" che permette di creare queste istanze

Oltre il modello procedurale

- Tutti questi meccanismi permettono di lavorare in maniera modulare e di costruire applicazioni più robuste e meglio organizzate
- Purtroppo i linguaggi procedurali non mettono a disposizione strumenti che obblighino o anche solo incoraggino a lavorare in questo modo
- Tutto è lasciato alla disciplina e all'esperienza di chi scrive le librerie e le applicazioni
- La separazione di fondo tra strutture dati e procedure rende difficoltoso operare correttamente
- Per gestire la complessità bisogna disporre di linguaggi che supportino i modo naturale uno stile di programmazione corretto
- Bisogna passare dal modello procedurale a quello orientato agli oggetti