Azzolini Riccardo 2021-02-22

Programmazione procedurale

1 Linguaggi di programmazione

Nel corso degli anni sono nati migliaia di linguaggi di programmazione: molti di questi sono stati usati da poche persone e poi abbandonati, mentre altri hanno avuto un maggior successo. Ciascun linguaggio ha delle particolari caratteristiche, che determinano come esso possa essere classificato secondo vari aspetti. In particolare, in questo corso saranno trattati i linguaggi C e C++, che sono:

- di **alto livello**: indipendenti dalla macchina sottostante;
- **portabili**: i programmi scritti in questi linguaggi possono girare su macchine diverse senza bisogno di apportare grosse modifiche al codice sorgente;
- **general-purpose**: possono essere utilizzati per risolvere problemi in vari ambiti (non solo in senso computazionale, ma anche per quanto riguarda gli aspetti pratici, poiché sostanzialmente tutti i linguaggi di programmazione hanno capacità di calcolo teoricamente equivalenti).

2 Programmazione procedurale

La **programmazione procedurale** è uno dei paradigmi di programmazione più noti e storicamente importanti. Esso è un caso particolare di **programmazione imperativa**, un paradigma più generale che vede una computazione (finalizzata a risolvere un determinato problema) come una sequenza di istruzioni (azioni, "ordini"). In particolare, la programmazione procedurale divide tale sequenza di istruzioni in delle **procedure** e **funzioni** che si chiamano tra di loro, interagendo tramite la condivisione di alcuni dati (le variabili globali e il passaggio di parametri). La differenza tra procedure e funzioni è che:

- una funzione *restituisce* almeno *un valore* al chiamante quando la sua esecuzione termina;
- una procedura *non restituisce alcun valore*, quindi ha senso solo se ha dei *side effect*, cioè se modifica lo stato dell'ambiente in cui viene eseguita (mentre una funzione può aver senso anche se non ha side effect, perché comunque ha almeno lo scopo di calcolare il valore restituito).

Nella progettazione di un linguaggio di programmazione procedurale, ci sono diversi aspetti che possono variare, tra cui ad esempio:

- le regole per il passaggio dei parametri (per valore o per riferimento);
- le regole di scope, ovvero cosa è visibile e cosa no (variabili globali e locali, ecc.) all'interno di ciascun frammento di codice;
- la possibilità di annidamento (*nesting*) delle procedure/funzioni, cioè di definirle una all'interno dell'altra, invece che tutte allo stesso livello (*flat*).

Tali aspetti, così come quelli che verranno presentati in seguito, corrispondono a scelte progettuali per le quali non esiste un'opzione "migliore" delle altre: ogni scelta rende un linguaggio più adatto per determinati usi e meno adatto per altri.

2.1 Rappresentazione dei dati

A basso livello, esistono sostanzialmente solo i tipi di dati numerici. Tutti i tipi più complessi devono dunque essere definiti come astrazione sulla base dei tipi numerici.

In genere, i linguaggi procedurali prevedono una serie di *tipi predefiniti* che corrispondono più o meno ai tipi a basso livello (numeri interi e reali, caratteri, ecc.), e permettono al programmatore di definire tipi più complessi a partire da essi, tramite i *costruttori di tipo*, ma tendono a non supportare l'overloading degli operatori.

La dimensione e il layout in memoria delle strutture dati sono in genere fissate nella fase di compilazione (le strutture dati dinamiche vengono quindi implementate tramite puntatori e gruppi di celle di dimensione fissa), e l'allocazione della memoria (dinamica) viene di solito gestita esplicitamente.

2.2 Tipizzazione

Una variabile corrisponde a una zona in memoria, che comprende un determinato numero di bit. Il significato della variabile dipende allora da come tali bit sono interpretati, il che è a sua volta determinato dal **tipo** della variabile stessa. Il processo che assegna un tipo a una variabile è detto **tipizzazione**. In base a come esso avviene, si distinguono due categorie di linguaggi:

- linguaggi **tipizzati**: a ogni variabile viene associato un tipo fissato e non modificabile;
- linguaggi **non tipizzati**: il tipo di una variabile può cambiare durante l'esecuzione.

Più nel dettaglio, il processo di tipizzazione comprende due aspetti, *type binding* e *type checking*, che possono essere statici o dinamici. In generale, nel contesto dei linguaggi di programmazione, l'aggettivo "*statico*" si riferisce a qualcosa che accade in fase di compilazione, mentre "*dinamico*" indica ciò che avviene in fase di esecuzione. Realizzare una determinata funzionalità in modo dinamico può allora consentire una maggiore flessibilità, ma ha quasi sempre lo svantaggio di una minore efficienza, perché comporta lo svolgimento di operazioni aggiuntive ogni volta che il programma viene eseguito, invece che una volta sola quando esso viene compilato.

Il **type binding** determina quando viene fissato il vincolo sul tipo di una variabile.

- Nei linguaggi tipizzati è **statico**: il tipo di una variabile è fissato al momento della compilazione, e non può cambiare. I vantaggi sono una minore possibilità di errori in esecuzione e una maggiore efficienza in esecuzione (perché il programma compilato sa già come interpretare il contenuto della zona di memoria corrispondente a una variabile, senza bisogno di determinare "al momento" il tipo della variabile), mentre gli svantaggi sono una minore flessibilità a livello progettuale, un aumento del numero di variabili e una sintassi meno snella.[1]
- Nei linguaggi non tipizzati è **dinamico**: il tipo di ogni variabile viene fissato al momento dell'esecuzione, e può cambiare.

Il **type checking** consiste nel controllare i tipi delle espressioni per verificare che le operazioni di un programma siano applicate correttamente. Anch'esso è statico nei linguaggi tipizzati, che controllano la correttezza delle espressioni durante la compilazione, e dinamico nei linguaggi non tipizzati, che rimandano tali controlli alla fase di esecuzione.

[1] Alcuni linguaggi tipizzati sono in grado di determinare automaticamente i tipi delle variabili, senza bisogno che il programmatore li specifichi esplicitamente, permettendo così di evitare l'appesantimento della sintassi che il type binding statico tipicamente comporta.