Programmazione Avanzata

2024-2025

Introduzione

Il linguaggio di programmazione ideale facilita la scrittura di programmi succinti e chiari. Questo ne permette la comprensione, modifica e mantenimento durante l'intero ciclo di vita. Aiuterà inoltre i programmatori a gestire l'interazione tra le componenti di un sistema software complesso. A un software è richeisto di essere affidabile, manutenibile ed efficiente. Ad un linguaggio di programmazione si chiede di essere scrivibile, ovvero di permettere la stesura di una soluzione in modo non contorto; leggibile, ovvero di permettere di riconoscere la correttezza o gli errori direttamente dalla sintassi, senza eseguire; semplice, ovvero facile da apprendere e applicare; sicuro, ovvero contenere protezioni contro la scrittura di codice malevolo; robusto, ovvero resistente ad eventi indesiderati.

Agli inizi, la programmazione era effettuata direttamente in codice macchina per ottenere programmi piccoli ed efficienti. È negli anni '50 che emergono i primi due linguaggi, Fortran e Cobol. Il primo permette di scrivere programmi in forma matematica, il secondo è adatto all'uso bancario. Le necessità dei programmatori sono cambiate nel corso degli anni. Funzionalità e paradigmi un tempo considerati inefficienti, come la ricorsione e la programmazione ad oggetti, sono oggi diventati la norma. Le caratteristiche del linguaggio sono, in ogni caso, definite al punto d'incontro tra le necessità umane del programmatore e le necessità tecniche dell'architettura di Von Neumann della macchina sottostante. Possiamo distinguere diversi paradigmi. La programmazione procedurale sceglie la *routine* come unità base per la modularizzazione. La programmazione imperativa si basa su istruzioni, definite a passaggi, che modificano valori. La programmazione funzionale segue un approccio simile a quello matematico, basato su espressioni e funzioni. La programmazione a oggetti si basa sul concetto di classe come unità base. La programmazione *abstract data type* usa i tipi astratti come unità base.La programmazione dichiarativa cerca di definire il problema tramite regole, invece di descrivere i passaggi per trovare la soluzione.

Computabilità

Un programma per computer è interpretabile come funzione matematica dello stato della macchina prima dell'esecuzione e degli ingressi forniti dall'utente. Esso può implementare solo funzioni computabili, ovvero in grado di produrre un risultato. Questi può essere impossibile da raggiungere per errori nella funzione, oppure a causa di un tempo di esecuzione infinito. Alcune funzioni possono essere computabili in principio, ma non in pratica (se il tempo di computazione eccede limiti materiali). Si definisce *funzione parziale* una funzione definita solo per certi argomenti. Usando le definizioni matematiche:

- funzione computabile: $f: A \to B$ è un insieme di coppie $f \subseteq A \times B$ che soddisfano le seguenti condizioni:
  - $< x, y > \in f$ e $< x, z > \in f \ \to \ y = z$
  - $\forall x \in A, \quad \exists y \in B \ / \ < x, y > \in f$
- funzione parziale $f: A \to B$ è un insieme di coppie $f \subseteq A \times B$ che soddisfano la seguente condizione:
  - $< x, y > \in f$ e $zx, z > \in f \ \to \ y = z$

Possiamo fornire una definizione alternativa di computabilità. Una funzione è computabile se esiste un algoritmo che permetta di produrre il risultato desiderato per qualsiasi ingresso appartenente al dominio. Anche quando l'algoritmo esiste, la sua implementabilità dipende dal linguaggio di programmazione scelto. La classe delle funzioni computabili sui numeri naturali coincide con la classe delle funzioni parziali ricorsive. Questo perché la ricorsione è essenziale nella computazione, e perché la maggior parte delle funzioni è parziale.

Un'altra definizione di computabilità si basa sul concetto di macchina di Turing. Una macchina di Turing è un sistema bicomponente. Il primo elemento è un nastro, diviso in celle di memoria, sul quale sia possibile leggere, scrivere e muoversi di una cella per volta. Il secondo elemento è un controllore a stati finiti, che opera sul nastro per leggerlo, per scriverci o per muovervisi di una cella. Una funzione sui numeri naturali è computabile con un metodo efficace se e solo se è computabile da una macchina di Turing. Questo teorema è dimostrabile con tre dimostrazioni: Alonso Church, Alan Turing e Calcolo Lambda. Tutti i linguaggi di programmazione sono *Turing-complete*.

In una quarta definizione, la computabilità è riconducibile all'*halting problem*, che consiste nel determinare se un programma terminerà in corrispondenza di un certo ingresso. Possiamo associare il problema ad una funzione f_{halt} a

doppio ingresso (programma, input) che ritorna *halt* o $\neg halt$ se il programma termina o meno. Supponendo di avere un programma in grado di risolvere il problema, ovvero che abbia lo stesso output di f_{halt}, possiamo utilizzarlo per creare un programma che a volte non termina.

Compilatori, calcolo lambda, semantica denotativa, divisione dei linguaggi

Un compilatore traduce il programma in istruzioni macchina, mentre un interprete traduce ed esegue allo stesso tempo. Il compilatore è divisibile in componenti. Il *lexical analyzer* raggruppa le istruzioni in *token*. Il *syntax analyzer* o *parser* raggruppa i *token* in espressioni, *statement* e dichiarazioni, in vase a regole grammaticali. Il prodotto del *parser* è il *parse tree*, una struttura dati che rappresenta il programma. Il *semantic analyzer* applica regole e procedure aggiuntive in base al contesto delle espressioni, come ad esempio il *type checking*, producendo un *augmented parse tree*. L'*intermiediate code generator* produce una prima versione non ottimizzata del codice, in un formato chiamato *intermediate representation*. Il *code optimizer* elimina sottoespressioni, sostituisce variabili duplicate, rimuove istruzioni inutilizzate e rimpiazza le chiamate a funzioni brevi con il rispettivo codice (*inlining*) quando esso è più efficiente. Il *code generator* converte il codice intermedio in codice macchina per il *target* desiderato.

Distinguiamo tra sintassi (il testo di un programma) e semantica (la funzionalità che rappresenta). Una grammatica è composta da un simbolo iniziale, un insieme di simboli non temrinali, un insieme di terminali e un insieme di regole di produzione. Essa fornisce un metodo per definire un insieme infinito di espressioni. I non terminali sono i simboli utilizzati per esprimere la grammatica, mentre i terminali sono i simboli che appaiono nel linguaggio. Una grammatica è detta *ambigua* se la stessa espressione ammette più di un *parse tree*. I linguaggi umani uniscono ambiguità, frasi imperative, dichiarative, e interrogative. I linguaggi imperativi uniscono dichiarazioni e assegnamenti. Nella *semantica denotazionale* un programma è una funzione matematica da stato a stato. Lo stato è una funzione matematica che rappresenta i valori della memoria in un determinato stato dell'esecuzione di un programma.

Il lambda calculus è una notazione per descrivere la computazione, composta da tre parti. La prima è una notazione per descrivere le funzioni. La seconda è un meccansimo di prova per descrivere equazioni tra epsressioni. La terza è un insieme di regole di calcolo chiamate riduzioni. I due concetti principali del calcolo lambda sono le astrazioni lambda, per cui se M è un'espressione, $\lambda x.M$ è la funzione ottenuta trattando M come una funzione di x, e l'applicazione, ovvero l'anteposizione di un'espressione davanti ad un'altra per ottenere la composizione. Ad esempio, in $(\lambda x.x)M = M$, applichiamo una funzione identità all'espressione M. Un linguaggio di programmazione è interpretabile come un'applicazione del calcolo lambda, ovvero l'unione del calcolo lambda puro con tipi di dati aggiuntivi. Introduciamo ora il concetto di assegnazione delle variabili. Si dice libera una variabile che non è dichiarata nell'espressione (recuperare la definzione di algebra e logica). Nel calcolo lambda, al contrario di quanto avvenga nei linguaggi di programmazione procedurali come C, l'assegnamento di variabili non ha alcun effetto secondario ed è puramente funzionale.