Complementi di Programmazione

Tipizzazione

CdL Informatica - Università degli studi di Modena e Reggio Emilia
AA 2023/2024

Filippo Muzzini

Tipo di dato

- Un programma manipola **dati** attraverso **istruzioni**
- **Instruzioni**
  - **Indicano l'operazione da compiere su dati di un certo tipo**
- **Tipo**
  - **Indica cosa rappresenta il dato (e di conseguenza che operazioni sono permesse)**

Tipo di dato

La stessa istruzione (da un punto di vista sintattico) può assumere differenti semantiche in base al tipo di dato:

1+3

‘a’+’b’

Il tipo di dato è fondamentale per determinare il risultato di un’istruzione

Tipo di dato

All’interno del programma, un dato viene identificato con un **nome**.

Il programmatore utilizza il nome ma deve sapere:

- Il tipo (per non indicare operazioni sbagliate)
- Dimensione in memoria (per non saturarla)
- Scope e tempo di vita dell’entità (per sapere quando utilizzarla)

Obiettivi della tipizzazione

- L'uso della tipizzazione permette di scoprire codice privo di senso/illecito
- Nei linguaggi compilati viene fatto a compile time
- Si prevengono errori dovuti ai tipi
- Es. in C
- 3 / “Ciao” non è consentito (divisione tra intero e stringa)
- Il programma non compila

Alcune espressioni vengono però automaticamente convertite (casting). Es. ‘a’/5

Obiettivi della tipizzazione

Ottimizzazioni

- Alcune operazioni possono essere eseguite in maniera più efficiente
  - Es. x*2 può essere eseguita con uno shift di bit
- Per eseguire tali ottimizzazioni è necessario che il compilatore/interprete sappiano il tipo di dato

Obiettivi della tipizzazione

Astrazione

- La tipizzazione è utile anche al programmatore
- Non deve preoccuparsi di come sia rappresentato il dato (se non in casi estremi)
  - Non deve preoccuparsi che una stringa in realtà siano byte
  - Non deve preoccuparsi se la macchina usi la rappresentazione little/big endian

Obiettivi della tipizzazione

Interfacce

- La tipizzazione è utile per definire interfacce
  - Gli argomenti tipizzati sono più esplicativi
  - Prevengono il passaggio di dati sbagliati

Type Check

- L'uso della tipizzazione permette di scoprire codice privo di senso/illecito
- Nei linguaggi compilati viene fatto a compile time
- Si prevengono errori dovuti ai tipi
- Es. in C
- 3 / “Ciao” non è consentito (divisione tra intero e stringa)
- Il programma non compila

Alcune espressioni vengono però automaticamente convertite (casting). Es. ‘a’/5

E nei linguaggi dinamici?

Pro Linguaggi Dinamici

- Tipizzazione dinamica: In un linguaggio dinamico, non è necessario dichiarare esplicitamente il tipo di una variabile durante la sua creazione. Il tipo di una variabile può cambiare durante l'esecuzione del programma.

Type Check - Linguaggi Dinamici

- Avviene a runtime (non è possibile determinare a priori il tipo di dato)
- Più flessibile
  - Minor gestione di tutti i casi possibili
- Maggiore overhead
- Possibilità di errori a runtime
- Possibilità di comportamenti non previsti
  - Necessità di maggior controlli
  - Necessità di gestioni degli errori

Type Check - Recap

- Statico
  - Identifica errori a tempo di compilazione
  - Previene errori a runtime
  - Più performante
- Dinamico
  - Più flessibile: costrutti illegali in linguaggi statici (ES. y=5; y=’ciao’)
  - Più rapida prototipazione

Tipizzazione forte

l linguaggio di programmazione impone rigorosamente regole sulla conversione dei tipi di dati e sulla compatibilità dei tipi.

- Conversioni esplicite
- Operazioni solo tra tipi compatibili
- Obbligo di dichiarare il tipo delle variabili

Tipizzazione debole

il linguaggio può consentire conversioni implicitamente tra tipi di dati e può essere più permissivo nella gestione dei tipi.

- Conversioni implicite
- Operazioni permesse tra tipi incongruenti
- Le variabili possono non avere un tipo esplicito

Tipizzazione debole

il linguaggio può consentire conversioni implicitamente tra tipi di dati e può essere più permissivo nella gestione dei tipi.

- Conversioni implicite
- **Operazioni permesse tra tipi incongruenti**
- Le variabili possono non avere un tipo esplicito

a = 3; b = ‘58’;

a+b = ?

Tipizzazione debole

il linguaggio può consentire conversioni implicitamente tra tipi di dati e può essere più permissivo nella gestione dei tipi.

- Conversioni implicite
- **Operazioni permesse tra tipi incongruenti**
- Le variabili possono non avere un tipo esplicito

a = 3; b = ‘58’;

a+b = ?

In C, b è un puntatore -> aritmetica dei puntatori

Tipizzazione debole

il linguaggio può consentire conversioni implicitamente tra tipi di dati e può essere più permissivo nella gestione dei tipi.

- Conversioni implicite
- **Operazioni permesse tra tipi incongruenti**
- Le variabili possono non avere un tipo esplicito

a = 3; b = ‘58’;

a+b = ?

In Java, a è convertito in stringa -> ‘358’

Tipizzazione debole

il linguaggio può consentire conversioni implicitamente tra tipi di dati e può essere più permissivo nella gestione dei tipi.

- Conversioni implicite
- **Operazioni permesse tra tipi incongruenti**
- Le variabili possono non avere un tipo esplicito

a = 3; b = ‘58’;

a+b = ?

In Perl, b è convertito in intero -> 61

Tipizzazione debole

il linguaggio può consentire conversioni implicitamente tra tipi di dati e può essere più permissivo nella gestione dei tipi.

- Conversioni implicite
- **Operazioni permesse tra tipi incongruenti**
- Le variabili possono non avere un tipo esplicito

a = 3; b = ‘58’;

a+b = ?

In Python -> Errore!

Tipizzazione safe

Un linguaggio di programmazione è considerato adottare una tipizzazione safe dei dati se impedisce che un'operazione di casting implicito causi un crash.

a = 3; b = ‘58’;

a+b = ?

In Perl, b è convertito in intero -> 61

Tipizzazione unsafe

Un linguaggio di programmazione è considerato adottare una tipizzazione unsafe dei dati se **non** impedisce che un'operazione di casting implicito causi un crash.

a = 3; b = ‘58’;

a+b = ?

In Python -> Errore!

a = 3; b = ‘58’;

a+b = ?

In C, b è un puntatore -> aritmetica dei puntatori (fuori range)