19. Puntatori

Corso di Algoritmi e Linguaggi di Programmazione Python/C

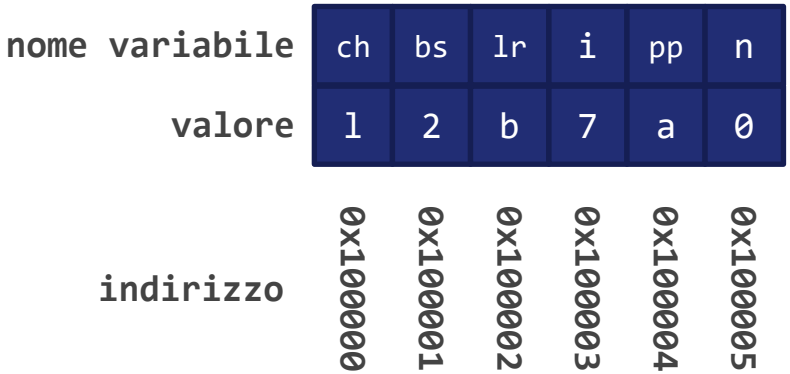
Angelo Cardellicchio
angelo.cardellicchio@stiima.cnr.it

Outline

- Il concetto di puntatore
- L’operatore **&**
- L’operatore di deferenziazione
- Passaggio per valore e per reference
- Puntatore come argomento di funzione
- Puntatore a **`void`**
- Puntatori ed array
- Puntatori e funzioni

Il concetto di puntatore

- I puntatori contengono al loro interno l'indirizzo di un'altra variabile.
  - *Ogni variabile ha un indirizzo in memoria.*
  - ***Il puntatore indica l'indirizzo!***
- I puntatori sono quindi degli interi.
  - *La rappresentazione dell'indirizzo è in esadecimale.*
- Sono possibili le operazioni aritmetiche.



L’operatore &

- L’operatore **&** restituisce l’indirizzo di una variabile.
  - *Tornando all’esempio precedente:*

    ```
    &ch == 0x100000
    ```

- Da qui, discende la modalità con cui si dichiara un puntatore:

  ```
  char* pointer_to_ch = &ch
  ```

- La notazione **`tipo*`** indica che stiamo definendo un puntatore a quello specifico **`tipo`**.

L’operatore di deferenziazione

- Oltre ad indicare la notazione con cui si contraddistinguono gli operatori, `*` è anche un operatore.
  - *Applicandolo ad un puntatore, possiamo avere il valore associato alla variabile puntata.*
  - *Tornando all’esempio precedente:*

    ```
    *pointer_to_ch == 1
    ```
- Questa operazione è chiamata **deferenziazione**.

Passaggio per valore e per reference

- È possibile passare una variabile ad una funzione in due modi.
- Il passaggio per **valore** implica che viene creata una **copia** della variabile, ed è questa copia ad essere mandata in input alla funzione.
  - *La variabile originaria non viene modificata.*
- Il passaggio per **reference** implica che viene passato un **riferimento** alla variabile in input alla funzione.
  - *La variabile originaria sarà modificata.*
- In C, gli argomenti sono sempre passati per valore.

Puntatore come argomento di funzione

- È possibile passare un puntatore come argomento ad una funzione.
- Il passaggio avverrà per valore, quindi sarà creata una **copia** del puntatore.
- Ciò implica che ogni operazione (anche aritmetica) effettuata sul puntatore avrà validità esclusivamente all'interno della funzione.
- Tuttavia, la **variabile puntata** rimane **accessibile e modificabile** all'interno della funzione!
- Si parla quindi (impropriamente) di passaggio **per indirizzo**.

Puntatori a `void`

- Creiamo un puntatore a **`void`** quando non conosciamo il tipo della variabile puntata a priori.

```
void* puntatore_variabile;
```

- Il puntatore a **`void`** è utile come **placeholder**.
  - *Non può essere infatti dereferenziato.*
  - *È opportuno quindi effettuare un cast al giusto tipo di puntatore quando il tipo della variabile viene definito.*

```
char* pointer_to_ch =
        (char*) pointer_to_generic_variable;
```

Puntatori ed array

- **Gli array sono un tipo di puntatore**.
- Dichiarare un array significa allocare memoria per l'array e per l'area puntata.
- Inoltre, il puntatore associato all'array è dichiarato **`const`**.
  - *Ciò implica che l'indirizzo dell'array non è modificabile, e che questo non possa essere usato come **l-value**.*
  - *Quindi la seguente assegnazione darà un errore:*

```
int lista[3];
lista = { 1, 2, 3 }; // Errore, un array non è un l-value!
```

Puntatori e funzioni

- Una funzione può avere un puntatore come valore di ritorno.
- Questo è di solito passato per valore.
- Ciò però non è garantito per la variabile puntata, che deve avere un ambito adeguato (quindi non locale alla funzione) o essere dichiarata `static`.
- Questa soluzione si usa spesso quando una funzione deve restituire una stringa.

Domande?

42