24. Puntatori

Corso di Informatica

Programmazione

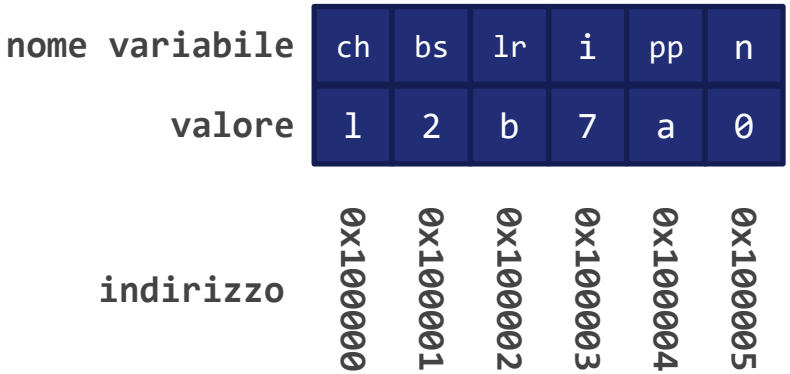
Corso di Laurea in Matematica (D.M. 270/04) - A.A. 2020/2021
Angelo Cardellicchio
angelo.cardellicchio@uniba.it
23/11/2020

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

- È un **tipo di dato associato ad una variabile che contiene l'indirizzo di memoria di un'altra variabile**
  - *Nella figura si nota come ogni **variabile** abbia un **valore** associato, il quale è salvato in una certo **indirizzo** della memoria*
  - ***Il puntatore indica l'indirizzo!***
- I puntatori sono quindi degli interi
  - *(ma sono rappresentati in esadecimale!)*
- Valgono le funzioni aritmetiche



L’operatore &

- L’operatore **&** restituisce l’indirizzo di una variabile.
  - *Utilizzandolo, è possibile ottenere un valore associabile ad un puntatore!*
  - *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`**.
  - *Ad esempio,* **`char*`** *è un puntatore a* **`char`**, *mentre* **`int*`** *è un puntatore ad* **`int`**

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 == l
    ```
- Quindi:

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

Intermezzo

- ***Esercizio 1**: scriviamo due funzioni,* **`mostra_puntatore_intero`** *e* **`mostra_puntatore_decimale`***, che accettano rispettivamente un valore intero ed un valore double. Le due funzioni non restituiscono alcun valore (quindi usano il tipo di ritorno* **`void`***), e stampano a schermo l'indirizzo di memoria associato alla variabile passata come parametro. Come* ***format specifier*** *si utilizzi il valore* **%p**.

Passaggio per valore e per reference

- In C (ed in ogni altro linguaggio di programmazione) è possibile passare una variabile ad una funzione in due modi
- Il primo modo è per **valore**: ciò significa che viene creata una **copia** della variabile, ed è questa copia ad essere mandata in input alla funzione
  - *Ovviamente, la variabile originaria non viene modificata!*
- Il secondo modo è per **reference**: ciò implica che viene passato un **riferimento** alla variabile in input alla funzione
  - *Ciò implica che la variabile originaria sarà modificata*
- In C, gli argomenti sono sempre passato per valore

Puntatore come argomento di funzione

- È possibile quindi 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**
  - *In realtà, l'argomento passato non è mai la variabile puntata, ma il puntatore alla stessa, e quindi vale sempre il passaggio per valore*

Intermezzo

- ***Esercizio 2**: scriviamo altre due funzioni, **`deferenzia_compara_intero`** e **`deferenzia_compara_decimale`**, ognuna delle quali accetta due parametri in ingresso, ovvero una variabile ed un puntatore alla stessa. La prima funzione tratta valori interi, la seconda reali. Le funzioni devono verificare la correttezza della dereferenziazione, e restituiscano il valore della variabile deferenziata.*
- ***Esercizio 3**: scriviamo una funzione, **`restituisci_puntatore`**, che accetta come parametro un intero, e restituisce il puntatore allo stesso. Verificare se l'indirizzo associato al puntatore restituito è lo stesso ottenuto utilizzando l'operatore **`&`** all'interno della funzione chiamante (ad esempio, il **`main`**).*

Puntatori a `void`

- A volte è necessario creare un puntatore prima che il programma «conosca» il tipo della variabile puntata. In questo caso, usiamo un puntatore a **`void`**:

```
void* pointer_to_generic_variable;
```

- 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;
```

Intermezzo

- ***Esercizio 4****: scriviamo la funzione* **`puntatore_a_void`** *che accetta un valore di tipo* **`char`** *e fa in modo tale che venga puntato da un puntatore a* **`void`**. *Utilizzare le opportune operazioni di cast per mostrare a schermo il valore della variabile passato alla funzione.*

Puntatori ed array

- **Gli array sono un tipo di puntatore**
- Dichiarare un array significa allocare memoria per l'array e per l'area puntata
  - *Ecco perché è necessario definirne a priori la lunghezza!*
- Inoltre, il puntatore associato all'array è dichiarato **`const`**
  - *Ciò implica che l'indirizzo dell'array non è modificabile, e questo fa sì che l'array 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!
```

- Altra differenza sta nel fatto che l'area puntata da un array può essere inizializzata, mentre in un puntatore questo non è ammesso
- Eccezione notevole sono i puntatori a **`char`** quando l'area puntata è una stringa literal

Puntatori e funzioni

- Una funzione può avere un puntatore come valore di ritorno
- In generale, il valore di ritorno è passato **by value**: ciò garantisce che il puntatore generato all'interno della funzione sia restituito con il giusto 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.
- Esistono anche i puntatori a funzione, che vedremo più avanti.

Finale

- ***Esercizio 5:*** *scriviamo una funzione* **`mio_nome`** *che restituisca una stringa rappresentativa dei nostri nomi e cognomi. Questa funzione dovrà essere chiamata direttamente dal* **`main`**.

Domande?

42