Procedure

Una *procedura* permette di

- *dare un nome* a una istruzione
- rendendola *parametrica*
- *non denota un valore, quindi non c'è tipo di ritorno*
  → **`void`**

```
void p(int x) {
float y = x * sin(0.75);
}
```

In C, una procedura ha la stessa struttura di una funzione, salvo il ***tipo di ritorno*** che è **`void`**

Procedure

L'istruzione *return* provoca solo la restituzione del controllo al chiamante e *non è seguita da* una espressione da restituire -> *non è necessaria* se la procedura termina "spontaneamente" a fine blocco

Nel caso di una procedura, non esistendo valore di ritorno, comunica con il chiamante solo:

- mediante ***parametri***
- mediante ***aree dati globali***

Occorre il ***passaggio per riferimento*** per fare cambiamenti permanenti ai dati del cliente

Passaggio dei parametri

In generale, un parametro può essere trasferito dal cliente al servitore:

- **per valore o copia** (*by value*)
  si trasferisce *il valore* del parametro attuale
- **per riferimento** (*by reference*)
  si trasferisce *un riferimento* al parametro attuale

Esempio: scambio variabili

Problema: scrivere una procedura che *scambi i valori di due variabili intere*

Specifica:

Dette A e B le due variabili, ci si può appoggiare a una *variabile ausiliaria T*, e svolgere lo scambio in *tre fasi*

```
void scambia(int a, int b) {
  int t;
  t = a; a = b; b = t;
  return; /* può essere omessa */
}
```

Esempio: scambio variabili

Il cliente invocherebbe quindi la procedura così:

```
main(){
   int y = 5, x = 33;
   scambia(x, y);
   /* ora dovrebbe essere
   x=5, y=33 ...
   MA NON È VERO
   */
}
```

Perché non funziona?

Esempio: scambio variabili

- La procedura ha *effettivamente scambiato* i valori di A e B *al suo interno*
- *ma questa modifica non si è propagata al chiamante*,
- perché sono state scambiate *le copie locali alla procedura,* ***non gli originali***
- al termine della procedura, le sue variabili locali *sono state distrutte →nulla è rimasto* del lavoro svolto dalla procedura

Passaggio per RIFERIMENTO

Il passaggio per riferimento (*by reference*)

- NON trasferisce ***una copia del valore*** del parametro attuale
- ***ma un riferimento al parametro****,* in modo da dare alla procedura (o funzione) *accesso diretto* al parametro in possesso del chiamante
- La procedura, quindi, ***accede direttamente*** al dato del chiamante *e* ***può modificarlo***

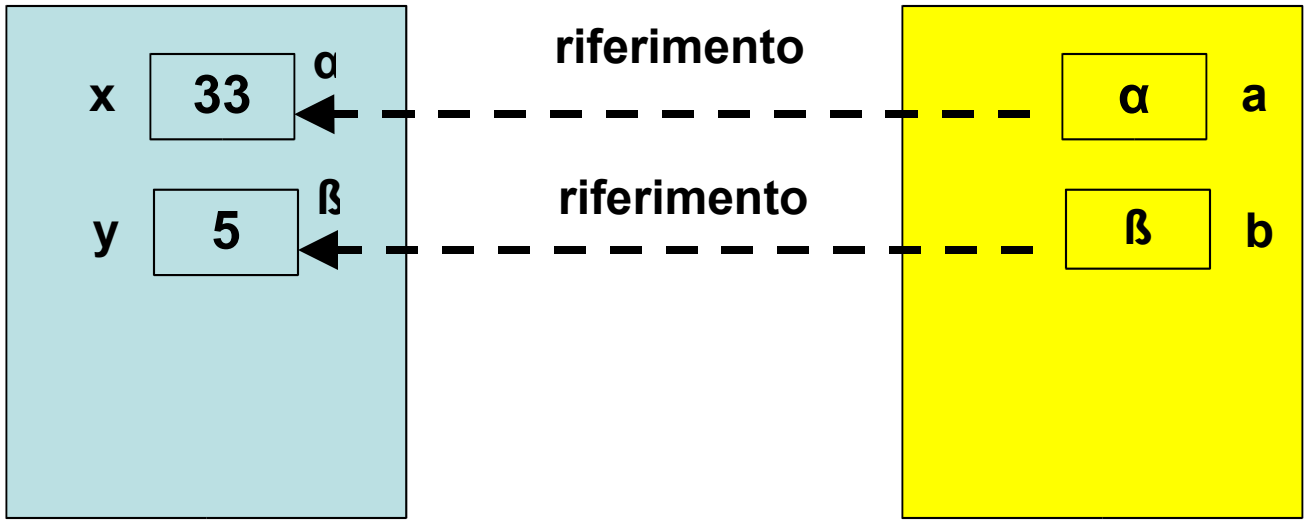
- Il linguaggio C ***NON*** supporta *direttamente* il ***passaggio per riferimento,*** occorre quindi ***costruirlo quando serve***

Passaggio per RIFERIMENTO

Si trasferisce *un riferimento* ai parametri attuali (cioè i loro indirizzi)

Ogni azione fatta su **a** e **b** *in realtà è fatta su **x** e **y*** nell'environment del chiamante

Quindi, scambiando a e b in realtà si scambiano x e y



Passaggio per RIFERIMENTO

Poiché passare un parametro per riferimento comporta la capacità di manipolare ***indirizzi di variabili***…

… gestire il passaggio per riferimento implica la capacità di *accedere*, *direttamente o indirettamente*, *agli indirizzi* delle variabili

Occorre essere capaci di:

- ***ricavare l'indirizzo*** di una variabile
- ***dereferenziare un indirizzo*** di variabile, ossia "recuperare" il valore dato l'indirizzo della variabile

- In C il ***programmatore deve conoscere gli indirizzi*** delle variabili e quindi accedere alla macchina sottostante

Indirizzamento e Deferencing

Il C offre a tale scopo *due operatori*, che consentono di:

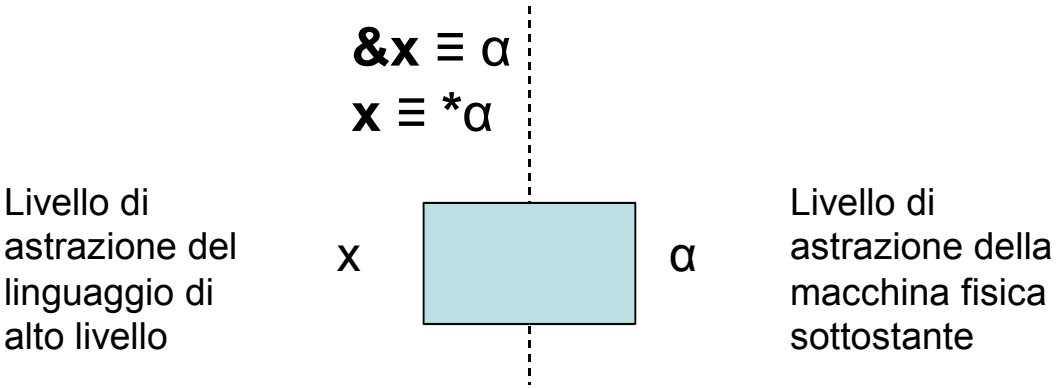
- ***ricavare l'indirizzo*** di una variabile:
  operatore estrazione di indirizzo &
- ***dereferenziare un indirizzo*** di variabile:
  operatore di dereferenziamento *

Se **x** è una variabile, **&x** denota l'*indirizzo in memoria* di tale variabile:

&x ≡ α

Se α è l'indirizzo di una variabile, *α denota *tale variabile*:

x ≡ *α



Puntatori

Un ***puntatore*** è il costrutto linguistico introdotto dal C (e da altri linguaggi) come *forma di accesso alla macchina sottostante* e in particolare agli ***indirizzi di variabili***

Un *tipo puntatore a T* è un tipo che denota l'indirizzo di memoria di una variabile di tipo T.
Un *puntatore a T* è una variabile di "*tipo puntatore a T"* che può contenere l'indirizzo di una variabile di tipo T

Definizione di una variabile puntatore:

<tipo> * <nomevariabile> ;

Esempi:

```
int   *p;
int*   p;
Int  * p;
```

Queste tre forme sono equivalenti, e definiscono p come *"puntatore a intero"*

Passaggio per Riferimento in C

Il chiamante deve *passare esplicitamente gli indirizzi*
La funzione (o procedura) deve *prevedere esplicitamente dei puntatori come parametri formali*

```
void scambia(int * a, int * b) {
        int t;
        t = *a; *a = *b; *b = t;
}
```

```
main(){
        int y=5, x=33;
        scambia(&x, &y);
}
```

Osservazione
Quando un puntatore è usato per realizzare il passaggio per riferimento, *la funzione* ***non dovrebbe mai alterare il valore del puntatore***
Quindi, se **a** e **b** sono due puntatori:

***a = *b** **SI**
a = b **NO**

In generale una funzione PUÒ modificare un puntatore, ma *non è opportuno che lo faccia se esso realizza un passaggio per riferimento*

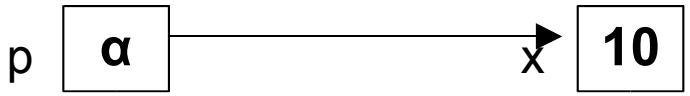
Puntatori

Un ***puntatore*** è una variabile *destinata a contenere* ***l'indirizzo*** *di un'altra variabile*

Vincolo di tipo: un puntatore a T può contenere solo l'indirizzo di variabili di tipo T

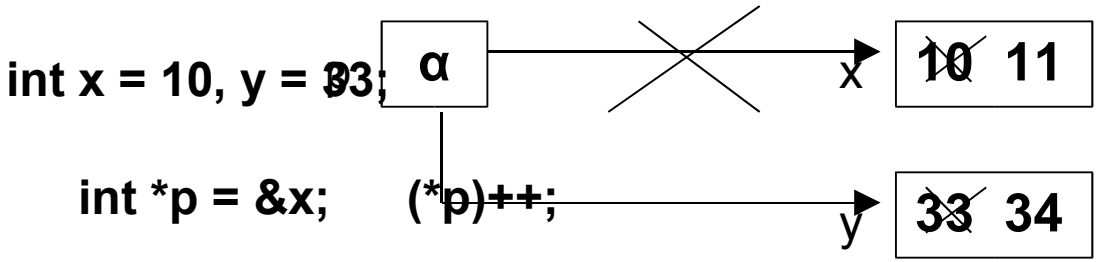
Esempio:

```
int x = 10;
int* p;
p = &x;
```



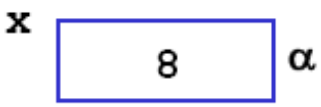
Da questo momento,***p** e **x** sono ***due modi alternativi per denotare la stessa variabile***

Un puntatore non è legato per sempre alla stessa variabile; può essere modificato

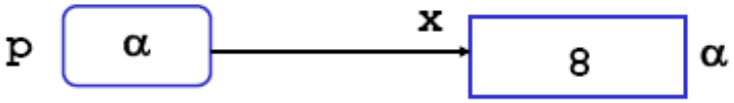


Esempi sui Puntatori

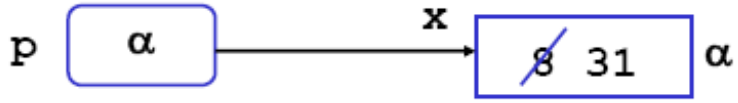
```
int  x = 8;
```



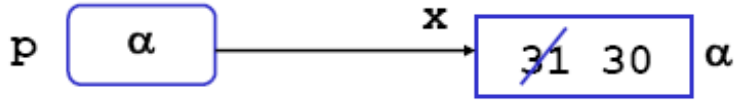

```
int* p = &x;
```




```
*p = 31;
```




```
x--;
```



Puntatore a tipo T

- Un puntatore a T può contenere *solo l'indirizzo di variabili di tipo T:* ***puntatori a tipi diversi sono incompatibili tra loro***

- Esempio:

```
int x=8, *p; float *q;
p = &x; /* OK */
q = p; /* NO! */
```

MOTIVO: il tipo *del puntatore* serve per *dedurre il tipo dell'oggetto puntato*, che è una ***informazione indispensabile per effettuare il dereferencing***

Esempio: scambia

Variazione dall'esempio precedente: i puntatori sono memorizzati in **px** e **py** prima di passarli alla procedura

```
void scambia(int* pa, int* pb) {
    int t;
    t = *pa; *pa = *pb; *pb = t;
}

main(){
    int y = 5, x = 33;
    int *py = &y, *px = &x;
    scambia(px, py);
}
```

Variabili Globali

- Una procedura può anche comunicare con il cliente ***mediante aree dati globali:*** ad esempio, ***variabili globali***

Le *variabili globali in C*:

- sono allocate ***nell'area dati globale*** (fuori da ogni funzione)
- esistono ***prima*** della chiamata del ***main***
- sono ***inizializzate automaticamente a 0*** salvo diversa
- indicazione
- possono essere *nascoste* in una funzione da una variabile locale omonima

Variabili Globali: esempio

- **Esempio:** Divisione intera x/y con calcolo di quoziente e resto. Occorre calcolare *due* valori che supponiamo di mettere in due variabili globali

int quoziente, int resto;

Le variabili globali **quoziente** e **resto** sono visibili in tutti i blocchi

```
void dividi(int x, int y) {
    resto = x % y; quoziente = x/y;
}
main(){
    dividi(33, 6);
    printf("%d%d", quoziente, resto);
}
```

Il risultato è disponibile per il chiamante nelle variabili globali **quoziente** e **resto**

Esempio: soluzione alternativa

- **Esempio:** Con il passaggio dei parametri per indirizzo avremmo il seguente codice

```
void dividi(int x, int y, int* quoziente, int* resto) {
    *resto = x%y; *quoziente = x/y;
}
main(){
     int k = 33, h = 6, quoz, rest;
     int *pq = &quoz, *pr = &rest;
     dividi(33, 6, pq, pr);
     printf("%d%d", quoz, rest);
}
```

Esercizi

Esercizio 1

Scrivere il codice della procedura “aggiungi” (di interfaccia data) che aggiunge a somma il valore di addendo.

void aggiungi(int *somma, int addendo)

Esercizio 2

Scrivere il codice della procedura “calcolaProvvigione” (di interfaccia data)

calcola_provvigione(int prezzo, int * provvigione)

dove:

prezzo: è un prezzo singolo

provvigione: è il puntatore a intero che memorizza il 15% di provvigione sul singolo prezzo fornito