

# Ambienti e sottoprogrammi

Andrea Marin

Università Ca' Foscari Venezia  
Laurea in Informatica  
Corso di Programmazione

a.a. 2015/2016

# Motivazioni

- ▶ Alcuni problemi si presentano frequentemente durante lo sviluppo di un programma
- ▶ Esempi:
  - ▶ Un programma che consente di fare operazioni con le frazioni userà a di frequente le funzioni mcm e mcd
  - ▶ Riscrivere ogni volta il codice per calcolarlo ha molti inconvenienti:
    - ▶ Scarsa leggibilità
    - ▶ Difficoltà nel mantenere il codice
    - ▶ Cattiva progettazione (monolitica vs. modulare)



# Ambienti e visibilità

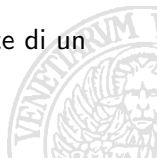
## Ambiente

L'ambiente è l'insieme di tutte le associazioni tra identificatori e locazioni di memoria.

## Visibilità

La visibilità (*scope*) di un identificatore è l'insieme delle posizioni nel codice dove quell'identificatore può o essere utilizzato.

- Le dichiarazioni hanno lo scopo di modificare l'ambiente di un programma



# Variabili globali e locali /1

- ▶ Le variabili dichiarate al di fuori di qualsiasi blocco (non racchiuse tra graffe) prendono il nome di variabili globali
- ▶ Le variabili dichiarate all'interno di un blocco prendono il nome di variabili locali o variabili **automatiche**



# Ambienti e variabili globali e locali

- ▶ Le variabili locali vengono allocate dinamicamente durante l'esecuzione del programma mentre quelle globali sono allocate prima dell'esecuzione della prima istruzione e rimangono allocate fino al termine dell'esecuzione del programma
- ▶ Le variabili locali vengono deallocate non appena termina il blocco in cui sono state dichiarate



## Regole di visibilità per variabili

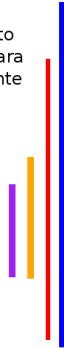
- ▶ Una variabile globale è visibile dal punto in cui è dichiarata in poi a meno che non venga nascosta
- ▶ Una variabile locale è visibile (a meno non venga nascosta) dal momento in cui è dichiarata fino al termine del blocco in cui essa è dichiarata
- ▶ Se una variabile locale prende lo stesso nome di una variabile precedentemente dichiarata, l'identificatore farà riferimento all'ultima dichiarazione *nascondendo* precedenti dichiarazioni



# Cosa stampa?

```
int x;  
int main(){  
    int y;  
    x=3;  
    y=x+3;  
    {  
        int a;  
        int y;  
        a=6;  
        y=2*a;  
        x = y;  
    }  
    printf("%d %d", x, y);  
    return 0;  
}
```

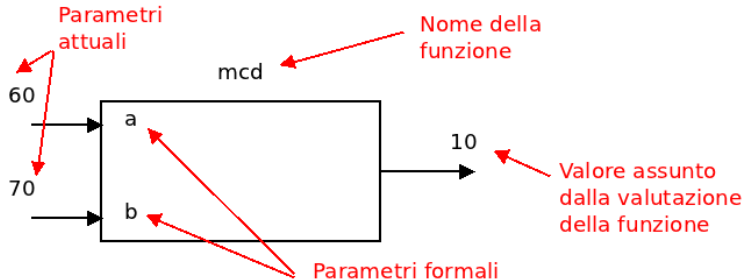
y fa riferimento  
a quello dichiara  
to più di recente



- ▶ Rappresenta graficamente la dinamica dell'ambiente del precedente programma
- ▶ Le barre a destra rappresentano la vita delle variabili



# Funzioni come black box



```
int mcd(int a, int b);
```

Firma della funzione





## Uso delle funzioni: MCD di 3 numeri $>0$

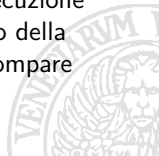
```
#include<stdio.h>
/* Dichiarazione della funzione*/
int mcd(int a, int b);

int main() {
    int x, y, z;
    int m1 , m2;
    scanf ( "%d" , &x ) ;
    scanf ( "%d" , &y ) ;
    scanf ( "%d" , &z ) ;
    m1 = mcd ( x , y ) ;
    m2 = mcd ( m1 , z ) ;
    printf ( "Il mcd e %d" , m2) ;
    return 0;
}
```



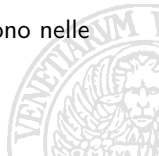
# Chiamata ad una funzione

- ▶ Quando in un'espressione compare il nome della funzione essa può essere valutata
  1. Si istanzia l'ambiente della funzione
    - ▶ Parametri formali
    - ▶ Variabili locali
  2. I valori dei parametri attuali (che sono espressioni) vengono **copiati** nei parametri formali appena allocati
  3. Si esegue il corpo della funzione
  4. L'istruzione `return exp` causa la terminazione dell'esecuzione della funzione e il valore di `exp` viene sostituito al posto della chiamata a funzione nell'espressione dove la funzione compare



## Alcune osservazioni

- ▶ Tipi dei parametri attuali e formali (condizioni sufficienti per la compilazione)
  - ▶ L'espressione che fa da parametro attuale ha lo stesso tipo del parametro formale
  - ▶ Il valore restituito dal return ha lo stesso tipo specificato come tipo restituito dalla funzione
  - ▶ L'abbinamento tra parametri formali ed attuali avviene per posizione e non per nome
  - ▶ Le modifiche ai parametri formali fatte all'interno della funzione non si ripercuotono sulle variabili che compaiono nelle espressioni corrispondenti ai parametri attuali



# Passaggio per copia o per valore

- ▶ A differenza di altri linguaggi (es. C++, Pascal) il C implementa solo il passaggio per copia
  - ▶ Cioè i parametri attuali sono **espressioni** il cui **valore** viene **copiato** nelle locazioni riservate ai parametri formali allocati automaticamente ad ogni chiamata della funzione
- ▶ Altri linguaggi che implementano solo il passaggio per copia: Java
- ▶ Altro modo (non esiste in C) di passare i parametri è per indirizzo o referenza
  - ▶ In questo caso il parametro attuale è una variabile e il parametro formale è un **alias** per quella locazione



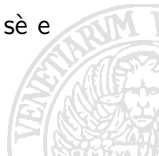
## Esempio di definizione di una funzione

```
/* calcola mcd tra a e b */  
/* Condizioni : a>0 e b>0 */  
int mcd(int a, int b) {  
    while ( a != b ) {  
        if (a > b)  
            a = a-b;  
        else  
            b = b-a;  
    }  
    return a;  
}
```



# Funzioni che ritornano void

- ▶ Talvolta può essere utile creare delle funzioni che non ritornano alcun valore
- ▶ Esempio: una funzione stampa il valore dei parametri in un certo formato
- ▶ In questi casi la funzione ritorna il tipo `void`
- ▶ Non serve l'istruzione `return` oppure deve comparire senza alcun valore
- ▶ La chiamata alla funzione comparire come istruzione a sè e non inserita in un'espressione



## Esempio

```
void stampa_ordinati(int a, int b) {  
    if (a>b)  
        printf("%d %d", a, b);  
    else  
        printf("%d %d", b, a);  
}
```

```
int main() {  
    /* leggi a e b */  
    stampa_ordinati(a,b);  
    return 0;  
}
```



# Errori comuni: da evitare!

- ▶ Creare una funzione con parametri formali e poi acquisirne il valore da standard input
  - ▶ Perché è sbagliato?
- ▶ Pensare che il valore dei parametri formali o delle variabili locali ad una funzione persista alla sua chiamata
  - ▶ Esempio: creare una variabili locale in una funzione che conta quante volte quella funzione è stata chiamata, perchè è sbagliato?





# Una questione di stile

- ▶ Input e Output da sottoprogrammi
  - ▶ È buono stile di programmazione distinguere le funzioni che fanno **solo** input/output da quelle che fanno computazioni
  - ▶ Noi ci atterremo rigorosamente a questo principio
- ▶ Quindi:
  - ▶ O una funzione fa input/output e fa solo quello (es. stampa un menù e fa scegliere un'opzione)
  - ▶ O una funzione calcola un risultato ma **non** lo stampa su standard output, ma lo restituisce come visto precedentemente
- ▶ Il principio è utile per il riutilizzo del codice



# Esercizi

- ▶ Scrivere una funzione `is_prime` che dato un intero positivo decida se è primo (le funzioni di decisione restituiscono un booleano (intero) che è `true` se la proprietà è verificata, `false` altrimenti)
  1. Dare la firma della funzione
  2. Pensa ad un uso della funzione
  3. Dai la definizione della funzione
- ▶ Scrivere la funzione `next_prime` che dato un numero  $n$  in ingresso restituisce il più piccolo numero primo maggiore strettamente di  $n$ .
- ▶ Scrivere un `main` di prova che stampi i primi 100 numeri primi

