Dipartimento di Informatica e Sistemistica
Antonio Ruberti

"Sapienza" Università di Roma

Funzioni

Corso di Fondamenti di Informatica

Laurea in Ingegneria Informatica

(Canale di Ingegneria delle Reti e dei Sistemi Informatici - Polo di Rieti)

Anno Accademico 2007/2008

Prof. Paolo Romano

Si ringrazia il Prof. Alberto Finzi per aver reso
disponibile il materiale didattico sul quale si basano queste slides

Funzioni

Per definire un programma complesso è necessario affrontarlo dividendolo in parti separate (*dividi ed impera*): modularizzazioni.

Modularizzazione basata su astrazione:
- Astrazioni sui Dati: tipi di dati astratti
- Astrazione funzionale: definizione di operazioni complesse come funzioni.

Noi trattiamo modularizzazione funzionale: nozione di funzione in C (già viste ed usate le *funzioni della libreria standard* del C).

Una funzione permette di definire una sorta di sottoprogramma con:
- parametri in ingresso;
- parametri in uscita;

Funzioni C

a. Un programma in C è un insieme di funzioni;

b. Ogni funzione prende in ingresso un insieme di argomenti e restituisce un valore.

c. Ci deve sempre essere una funzione principale `main()`

Fino ad ora la sola funzione `main()` e *funzioni di libreria* (`#include<stdio.h>`).

Vediamo di seguito:

a. Come si definiscono le funzioni (definizione di funzione)
b. Come si usano le funzioni (chiamata o attivazione di funzione).
In realtà questo si è già visto con le funzioni di libreria.

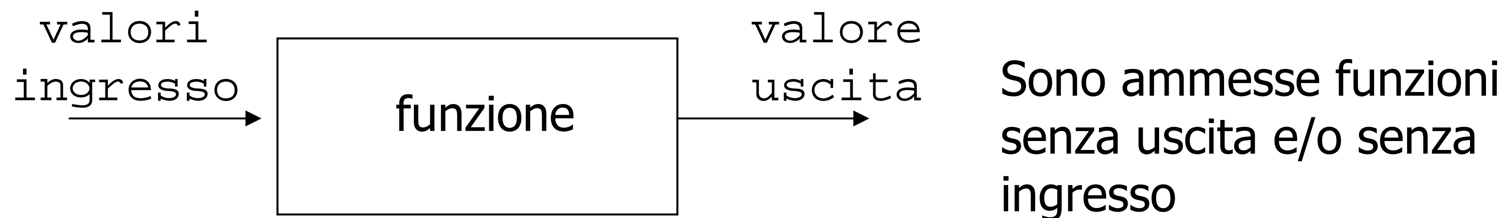
Esempio Programma con Funzione C

Programma con una funzione `square` per calcolare il quadrato di un numero.

```
#include<stdio.h>
int square(int); /*dichiarazione di funzione (o prototipo)*/

int main() {
   int x;

   for(x=1;x<=10;x++)
      printf(“il numero %d quadrato di %d \n”,square(x),x);

   return 0;

}
int square(int y) {/* definizione della funzione */
   return y * y;
}
```

Dichiarazione di Funzioni C

Come le variabili devono essere dichiarate prima di poter essere usate, anche le funzioni necessitano di una dichiarazione.



Sono ammesse funzioni senza uscita e/o senza ingresso

La dichiarazione di una funzione dovrà specificare:

a. Il nome della funzione: identificatore che ne permetterà l'utilizzo nel programma;

b. Gli argomenti (parametri) della funzione: quantità e tipo dei valori forniti in ingresso;

c. Il tipo del valore in uscita.

Dichiarazione di Funzioni C

Esempio: Funzione massimo tra due numeri:
- nome: `max`
- 2 argomenti di tipo `int`
- risutato: argomento di tipo `int`

Dichiarazione: `int max(int, int);`

Sintassi di dichiarazione di funzione (o prototipo):

```
identificatore-tipo identificatore(lista-tipo-parametri-formali);
```

- `identificatore-tipo` specifica il tipo del valore di ritorno (tipo del risultato della funzione), se manca viene assunto `int`

-`identificatore` specifica il nome della funzione (un qualunque identificatore C)

-`lista-tipo-parametri-formali` specifica il tipo dei dati passati alla funzione.
Lista è: `tipo1 nome1, …,tipoN nomeN` (parametri come variabili)
o solo: `tipo1, …, tipoN` (nella dichiarazione facoltativo nome parametro)

Esempi Dichiarazione di Funzioni C

Prototipi:

```
int fattoriale(int);        oppure
int fattoriale(int n);

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

double lit2euro(int);       oppure
double lit2euro(int lire);
```

Il nome del parametro può essere incluso nel prototipo per documentare meglio il programma, comunque sarà ignorato dal compilatore:

il prototipo associa il nome della funzione al suo tipo

$$f: tipo_1 \times tipo_2 \times \ldots \times tipo_n \rightarrow tipo$$

Tipo `void`

Esistono funzioni che non producono alcun effetto (es. funzione di stampa)

Il linguaggio C mette a disposizione un tipo speciale `void`

Ad es. `void f(int,int);`

è il prototipo di una funzione che non restituisce output

Es.
```
void f(int a, int b) {
        printf("%d",a*b);
     }
```

Il tipo `void` viene utilizzato anche per specificare l'assenza di argomenti:

le dichiarazioni `int f(void);` e `int f();` sono equivalenti

Definizione di Funzioni C

La definizione di una funzione specifica cosa fa una funzione e come lo fa: contiene la sequenza di istruzioni che dovrà essere eseguito per ottenere il valore risultato a partire dai dati in ingresso (una sorta di sottoprogramma).

La definizione di una funzione consta di due parti fondamentali:

1. Intestazione: simile alla dichiarazione (però necessaria la specifica dei parametri)

2. Blocco: del tutto simile al blocco già introdotto per la funzione `main()` e può contenere istruzioni e dichiarazioni di variabili (nota: non di funzioni!)

Esempi Definizione Funzioni C

Esempi Funzioni C

```
int max(int x, int y) {
    if (x >= y) return x;
    else return y;
}
```

Blocco: con **`return`** viene terminata ogni funzione C

Intestazione ⇨ `int max(int x, in y)`

nome

Parametri formali

Attenzione! Se si omette il tipo di un parametro questo è assunto `int`

`int max(double x, y)`

equivale a: `int max(double x, int y)`

e non a: `int max(double x, double y)`

Esercizio definire una funzione convertitore euro-lire: in ingresso prende un valore in lire, in uscita determina l'equivalente in euro

Definizione Funzioni C

Sintassi definizione funzione

```
intestazione blocco
```

- `blocco` è il corpo della funzione;
- `intestazione` è l'intestazione della funzione con la forma seguente:

```
identificatore-tipo identificatore ( lista-parametri-formali )
```

- `identificatore-tipo` specifica il tipo del valore di ritorno (tipo del risultato della funzione), se manca viene assunto `int`

-`identificatore` specifica il nome della funzione (un qualunque identificatore C)

-`lista-parametri-formali` specifica i dati passati alla funzione, è una lista di dichiarazioni di parametri (tipo nome) separate da virgola, ogni parametro è una variabile (la lista può essere vuota).

Esempio Definizione Funzioni C

```
/* trovare il maggiore di tre interi */
#include <stdio.h>

int maximum(int, int, int);

int main() {
   int a, b, c;

   printf("digita tre interi: ");
   scanf("%d%d%d",&a,&b,&c);
   printf("il massimo è: %d",maximum(a,b,c));
   return 0;
}
int  maximum(int x, int y, int z){        <---------  intestazione
   int max = x;
                                           <---------  blocco
   if (y > max) max = y;
   if (z > max) max = z;
   return max;
}
```

Attivazione di Funzioni

Le funzioni vengono chiamate (attivate) all'interno di funzioni

```
. . .
int valore, x, y;
 . . .
x = 1; y = 2;
valore = max(x,y);
 . . .
```

Sintassi Attivazione Funzione

```
identificatore (lista-parametri-attuali)
```

- `identificatore` è il nome della funzione
- `lista-parametri-attuali` è una lista di espressioni separate da virgola.

I parametri attuali devono corrispondere in numero e tipo ai paramenti formali.

Coercizione argomenti

Una chiamata di funzione che non corrisponda al prototipo provocherà un errore di sintassi.

Es.
```
int x;
x = maximum(1,3);
```

Errore a tempo di compilazione

In alcuni casi tollerate chiamate con tipi diversi: *coercizione degli argomenti*.

Es. `sqrt` ha come argomento un `double`, però:

```
int x = 4;
printf("%f",sqrt(x));
```
stampa `2.00…0`

Il compilatore *promuove* il valore intero `4` al valore `double 4,0` quindi la funzione viene chiamata correttamente.

Le conversioni avvengono secondo le *regole di promozione del C*

Attivazione di Funzioni

Semantica Attivazione Funzione di una funzione B in una funzione A

1. L'attivazione di una funzione è un'espressione.
2. L'attivazione di B da A determina:
   a. viene sospesa l'esecuzione di A e si passa ad eseguire le istruzioni di B (a partire dalla prima)
   b. quando termina l'esecuzione di B si torna ad eseguire le istruzioni di A

Prima di poter essere attivata una funzione deve essere definita (o dichiarata)

Quindi definita o dichiarata sopra.

Dove si definiscono le funzioni?

In C non si possono definire funzioni dentro funzioni.

Quindi fuori dalla funzione `main()` (sopra o sotto)

Esempio:

```
int max(int x, int y) {
   if (x >= y) return x;
   else return y;
}

int main {
   int a, b,c;

   scanf("%d, %d",&a,&b);
   max = max(a,b);
   return max;
}
```

definizione di funzione

chiamata di funzione

La funzione `max` è visibile nel `main`

Visibilità delle Funzioni

Prima di essere attivata una funzione deve essere definita o dichiarata:
il compilatore deve controllare che i parametri formali corrispondano (per numero e tipo) ai parametri attuali.

prima dell'attivazione deve essere conosciuto il prototipo

Esempio:

```
int main() {
    int a, b,c;

    scanf("%d%d",&a,&b);
    max = max(a,b);
    prinf("il max è %d",c);
    return 0;
}
int max(int x, int y) {
    if (x >= y) return x;
    else return y;
}
```

`max` non è visibile ed il compilatore C non può controllare i parametri

Visibilità delle Funzioni

Se la funzione non è definita almeno deve essere già dichiarata

```
int max(int, int);
int main () {
    int a, b,c;

    scanf("%d%d",&a,&b);
    c = max(a,b);
    prinf("il max è %d",c);
    return 0;
}
int max(int x, int y) {
    if (x >= y) return x;
    else return y
}
```

dichiarazione di funzione

Ordine dichiarazione delle funzioni

Ogni funzione deve essere stata dichiarata prima di essere usata.

E’ pratica comune specificare in quest’ordine:

1. Dichiarazione di tutte le funzioni con esclusione del `main`
2. Definizione di `main`
3. Definizione di tutte le altre funzioni

In questo modo ogni funzione e’ stata dichiarata prima di essere usata, e l’ordine diventa irrilevante!

Esempio:

```
int max(int,int);
int mcm(int,int);
int main(){ …}
int max(int a, int b){ … }
int mcm(int a, int b){…}
```

Nota: i file di intestazione della lib. standard (e.g. <stdio.h>) contengono i prototipi delle funzioni.

File header (o intestazioni)

Ogni libreria standard ha un corrispondente file header che contiene:

- definizioni di costanti
- definizioni di tipo
- dichiarazioni di tutte le funzioni della libreria

Esempi:

`<stdio.h>`	input/output
`<stdlib.h>`	memoria, numeri casuali, utilità generali
`<string.h>`	manipolazione stringhe
`<limits.h>`	limiti del sistema per valori interi
`<float.h>`	limiti del sistema per valori reali
`<math.h>`	funzioni matematiche

...

Possono essere scritte dal programmatore `"mialib.h"`

Funzioni della libreria matematica

Per utilizzarle occorre: `#include<math.h>`

Sia argomenti che valore di ritorno reali in doppia precisione, ovvero tipo `double` (non `float`)

Funzioni disponibili:

Funzione	Descrizione
`sqrt(x)`	radice quadrata
`exp(x)`	e^x
`log(x)`	logaritmo naturale
`log10(x)`	logaritmo base 10
`fabs(x)`	valore assoluto
`ceils(x)`	arrotonda all’intero più piccolo $> x$
`floor(x)`	arrotonda all’intero più grande $< x$
`pow(x,y)`	x^y
`fmod(x,y)`	resto di x/y (in virgola mobile)
`sin(x), cos(x), tan(x)`	trigonometriche (x in radianti)

Regole di visibilità

Ogni dichiarazione del linguaggio C ha un suo campo di visibilità.

Visibilità detta *nel file*

-L' identificatore è *dichiarato* all'esterno di ogni funzione:

L'identificatore sarà noto in tutte le funzioni la cui definizione è successiva

-L'identificatore è detto *identificatore globale*

```
#include<stdio.h>
int importo_in_lire;

main(){

  . . . .

  return 0;
}
```

Regole di visibilità

Visibilità detta *nel blocco*

Identificatore dichiarato in un blocco è limitata al blocco stesso

```
int a;
scanf("%d",&a);
if (a > 10) {
   int b = 10;
}
printf("%d",b);
```

Frammento di codice errato
(errore in fase di compilazione)

Le variabili dichiarate nel blocco istruzioni di una funzione sono dette variabili locali quindi riferibili solo nel corpo della funzione.

```
#include<stdio.h>
main(){
   int importo_in_lire;
   . . . .
   return 0;
}
```

variabile locale alla funzione `main`

Regole di visibilità

Visibilità detta *nel blocco:* Variabili locali

I *parametri formali* di una funzione sono considerati come variabili locali: visibili solo nel blocco della funzione.

Nota: non definire parametri con stesso identificatore; non definire variabile locale con nome di parametro:

```
int f(double x) {
    int x;
    . . .
  return 0;
}
```

Errore a tempo di compilazione

Regole di visibilità

Mascheramento

Il mascheramento si ha quando in un blocco si dichiara un identificatore già dichiarato fuori dal blocco ed in esso visibile.

La dichiarazione interna maschera quella esterna

Esempio:

```
#include<stdio.h>
int importo_in_lire;

main(){
  double importo_in_lire;
       . . . .
  return 0;
}
```

Regole di visibilità: mascheramento

```
#include <stdio.h>

int i = 1, j = 2;  /* variabili globali */

int f(){
  long i = 4;
  int k = 3;
  { /* blocco interno a f */
   float j = 8.5;
   printf("blocco di f: i = %d,  j = %g,  k = %d \n", i, j, k);

  }
  printf(“f: i = %d,  j = %d,  k = %d \n", i, j, k);
  return k;

} /* f */

int main(){
  j = f();
  printf("main: i = %d,  j = %d\n", i, j);

}
```

Tempo di vita di una variabile

Le variabili locali (le locazioni di memoria associate) vengono:
1. Create al momento dell'attivazione di una funzione
2. Distrutte al momento dell'uscita dall'attivazione

Segue che:
a. la funzione chiamante non può riferirsi ad una variabile locale alla chiamata
b. ad attivazioni successive di una stessa funzione corrispondono variabili (locazioni di memoria) diverse

Nota: il tempo di vita di una variabile è un concetto rilevante a run time.

Esempio

Scrivere un programma che stampa triangoli o rettangoli: l'utente può scegliere la figura da stampare e le sue dimensioni. La figura verrà visualizzata con linee di asterischi.

Algoritmo (in pseudocodice)

```
do
   leggi il tipo di figura
   switch su figura letta
      case è triangolo: stampa triangolo
      case è quadrato: stampa quadrato
      case nessuna: stampa saluto
while figura letta è diversa da nessuna
```

per stampa triangolo e stampa quadrato si utilizzano funzioni

Esempio

Algoritmo (in pseudocodice raffinato)
do
   stampa messaggio
   leggi un carattere
   switch su carattere letto
      case `‘t’`: stampa triangolo
      case `‘q’`: stampa quadrato
      case `‘f’`: stampa saluto
while carattere letto diverso da `‘f’`

Caratteri in C: si utilizza il tipo `char`

- Ogni carattere rappresentato dal suo codice
- Caratteri utilizzati come gli interi (carattere = codice)
- Carattere racchiuso tra apici, es. `‘a’ ‘B’ ‘d’`
- Specificatore di formato `%c`

Esempio

Funzione main:

```
int main(){
  char ch; int altezza;
  do {
    printf("\n digita \n");
    printf("  q: per stampare un quadrato\n");
    printf("  t: per stampare un triangolo\n");
    printf("  f: per terminare il programma:\n");
    scanf("%c", &ch);
    getchar();   /* serve per saltare il carattere '\n' */
    printf("\n");
    if (ch == ‘q’ || ch ==‘t’) {
       printf(“altezza?”);
       scanf(“%d”,&altezza); getchar();
    }
    switch(ch) {
        case 'q' : StampaQuadrato(altezza);  break;
        case 't' : StampaTriangolo(altezza); break;
        case 'f' : StampaSaluto();    break;
   }
} while (ch != 'f');
  return 0;
}
```

Esempio

Funzioni `StampaQuadrato`, `StampaTriangolo`:

```
void StampaQuadrato(int altezza) {
  int i, j;  /* i e j sono variabili locali */

  for(i = 1; i <= altezza; i++) {
    for(j = 1; j <= altezza; j++)
      printf("*");
    printf("\n");
  }
}

void StampaTriangolo(int altezza) {
  int i, j;  /* i e j sono variabili locali!!! */

  for(i = 1; i <= altezza; i++) {
    for(j = 1; j <= altezza - i ; j++)
      printf(" ");
    for(j = 1; j <= 2*i - 1 ; j++)
      printf("*");
    printf("\n");  }

}
```

Esempio

Ridefinizione `StampaQuadrato`, `StampaTriangolo` utilizzando una funzione `StampaRiga`:

```
void StampaRiga(int lunghezza, char carattere){
   int i;
   for(i = 1 ; i <= lunghezza ; i++) printf("%c",carattere);
}
void StampaQuadrato(int altezza) {
  int i, j;  /* i e j sono variabili locali */

  for(i = 1; i <= altezza; i++) {
    StampaRiga(altezza,'*');
    printf("\n");
  }
}

void StampaTriangolo(int altezza) {
  int i, j;  /* i e j sono variabili locali!!! */

  for(i = 1; i <= altezza; i++) {
    StampaRiga(altezza - i,' ');
    StampaRiga(2*i - 1,'*');
    printf("\n");  }

}
```

Esempio

Mettendo tutto insieme:

```
#include<stdio.h>
void StampaRiga(int,char);
void StampaQuadrato(int);
void StampaTriangolo(int);
void StampaSaluto();

void main(void){
   …
}
void StampaRiga(int lunghezza, char carattere){
  …
}
void StampaQuadrato(int altezza){
  …

}
void StampaTriangolo(int altezza){
  …
}
void StampaSaluto(){printf("ciao \n");}
```

Gestione della memoria a run time

Codice macchina e dati entrambi in RAM, ma in zone separate:

- Memoria per il codice macchina fissato a tempo di compilazione
- Memoria per dati locali alle funzioni (variabili e parametri) cresce e decresce dinamicamente durante l'esecuzione: gestito a pila.

Pila (stack): struttura dati con accesso LIFO: Last In First Out

Durante l'esecuzione del programma viene gestita la **pila dei record di attivazione** (RDA) in memoria centrale:

- per ogni attivazione di funzione viene allocato un RDA in cima allo stack
- al termine dell'attivazione della funzione il RDA viene rimosso

Record di Attivazione

Ogni **RDA** contiene:

a. Le *locazioni di memoria dei parametri formali* (se presenti)

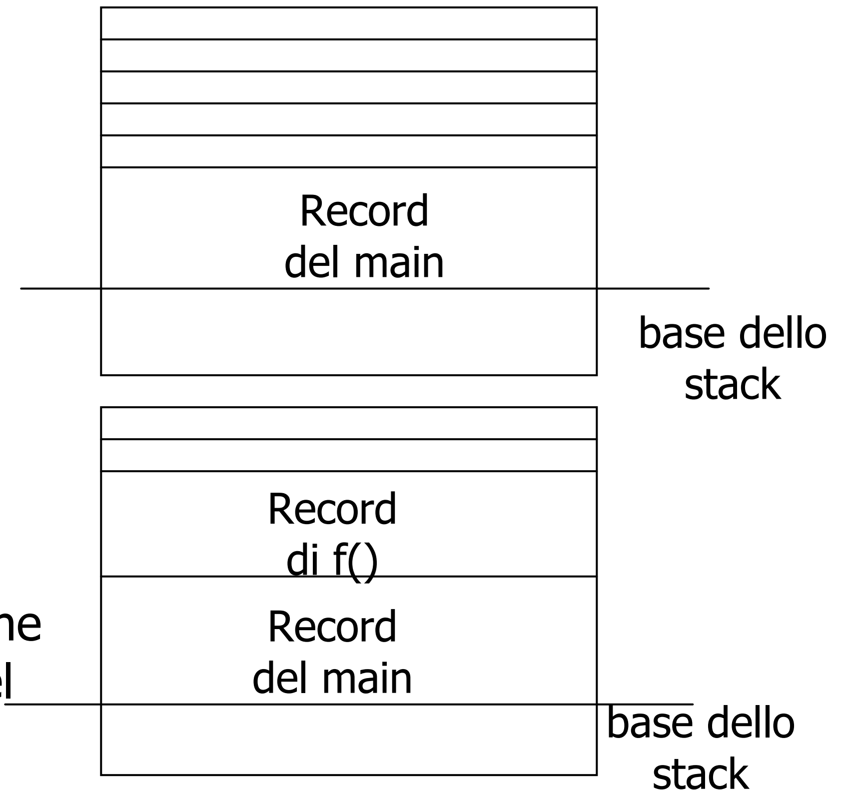
b. Le *locazioni di memoria per le variabili locali* (se presenti)

c. L'*indirizzo di ritorno*: indirizzo della prossima istruzione della funzione chiamante.

Quando un programma viene lanciato, l'unica funzione attiva è il `main`. Nello stack è riservato spazio per le variabili locali al `main`

Se durante l'esecuzione di `main()` viene chiamata la funzione `f()` viene allocato un nuovo *record di attivazione*

Il record di attivazione di f() contiene anche l' *indirizzo di ritorno (IDR)* all'istruzione del main successiva alla chiamata.



Esempio

```
void f(int q){
  printf("Sono f(). Il par. q vale %d\n", q);
  return;
}
void g (int p){
  int loc;
  printf("Sono g(). Il par. p vale %d\n", p);
  loc = 10;
  printf("Ora chiamo f(), con parametro attuale pari a %d\n", loc);
  f(loc);
  printf("Sono di nuovo g()\n");
  return;
}
int main(void){
  int s;
  printf("Sono main()\n");
  s = 5;
  printf("Ora chiamo g(), con parametro attuale pari a %d\n",s);
  g(s);
  printf("Sono di nuovo main()\n");
  return 0;
}
```

Esempio

Compilazione: Codice sorgente → Codice macchina

Codice macchina in memoria:

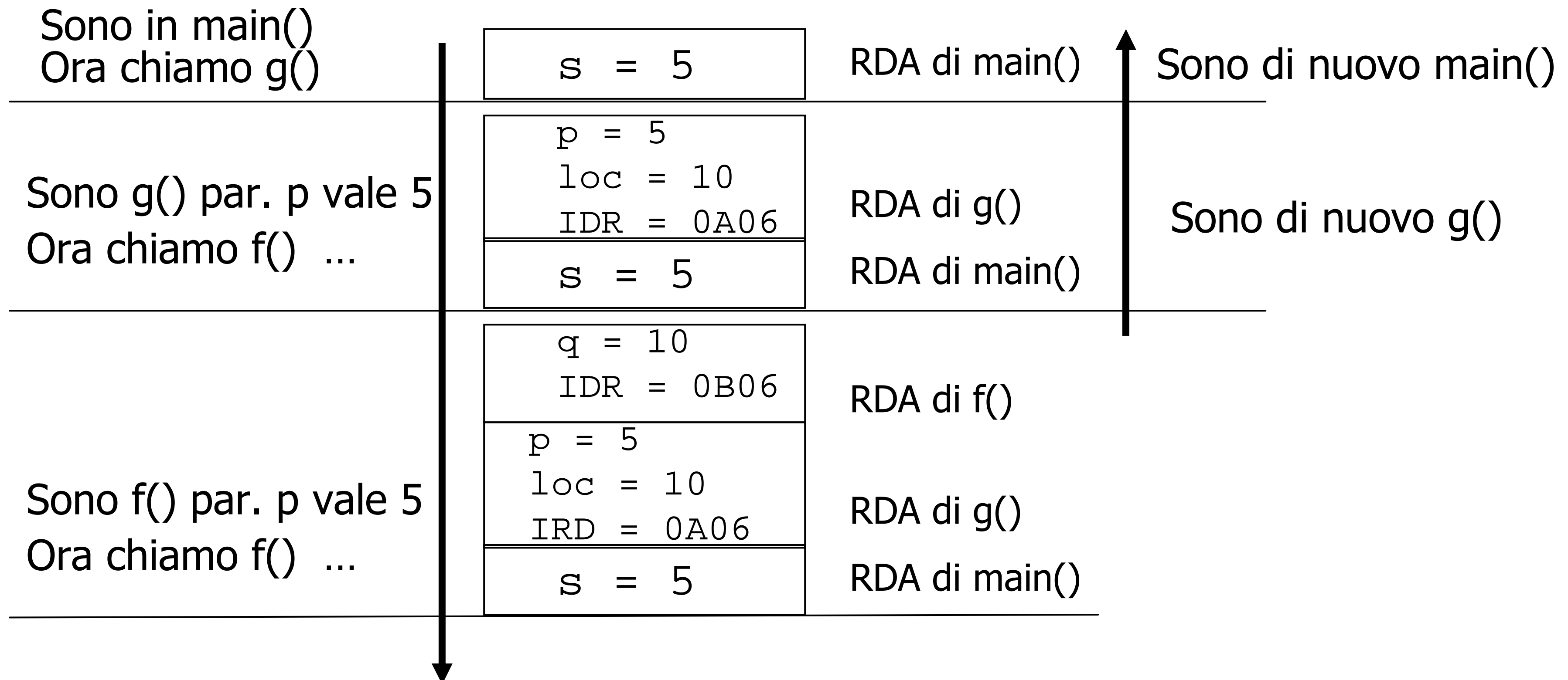
main()			g()			f()	
0A00	m1		0B00	a1		0C00	b1
0A01	m2		0B01	a2		0C01	b2
0A02	m3		0B02	a3		0C02	
0A04	m4		0B04	a4			
0A05	m5	Chiamata g(s)	0B05	a5	Chiamata f(s)		
0A06	m6		0B06	a6			
0A07	m7		0A07	a7			

Esecuzione programma con funzioni richiede:

1. Program Counter (PC)
2. Pila di RDA

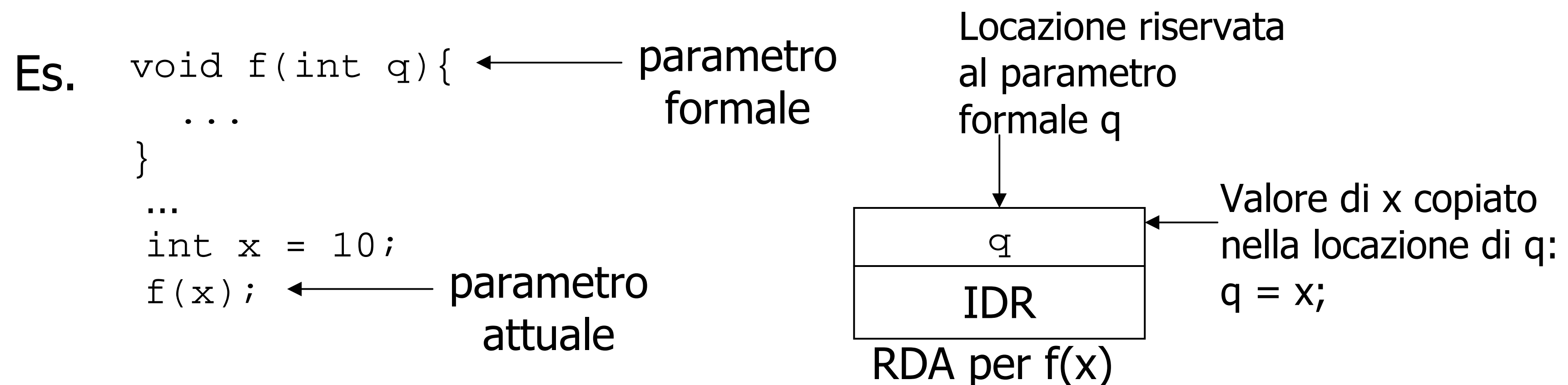
Esempio

Esecuzione del programma (IRD = Indirizzo Di Ritorno):



Passaggio di Parametri

Alla chiamata di funzione:
- viene creato nuovo RDA per la funzione
- vengono registrati i valori di ingresso nelle locazioni di memoria riservate ai *parametri formali* della funzione (*passaggio di parametri*)



In C passaggio per valore: il valore dei parametri attuali vengono copiati nelle locazioni riservate ai parametri formali.

Passaggio di Parametri

Limiti del *passaggio per valore*:

si vuole definire una funzione `void swap(int,int)` il cui effetto sia lo scambio di valore tra due variabili:

```
...
int x1= 0, x2 = 1;
swap(x1,x2);
printf("x1:%d x2:%d",x1,x2);
...
```

la seguente definizione non va bene!

```
    void swap(int a, int b) {
        int aux;
        aux = a;
        a = b;
        b = aux;
    }
```

Output: `x1:0 x2:1`. Lo scambio avviene su variabili locali e non ha effetto sui parametri attuali `x1,x2`!

Variabili Automatiche e Statiche

Tempo di vita di una variabile =
periodo in cui esiste la cella di memoria associata alla variabile =
periodo in cui esiste RDA relativo (variabili locali automatiche)

Una variabile **statica** esiste per tutto il tempo di esecuzione del programma:
- se è dichiarata all'esterno di ogni funzione
- se è locale ad una funzione e lo specificatore `static` precede la dichiarazione:
  Es. `void f(){ static int x; … }`

  - la var. viene inizializzata alla prima attivazione della funzione
  - conserva il valore tra attivazioni successive
  - è locale (visibile solo nella funz. in cui è dichiarata)

Esempio

```
#include<stdio.h>

void f(){
    int a = 0;
    static int b = 0;
    printf("\n a:%d b:%d",a,b);
    a++;
    b++;
}
main() {
    int j;

    for (j= 0; j <3; j++)f();
}
```

Il programma visualizza:

```
a:0 b:0
a:0 b:1
a:0 b:2
```