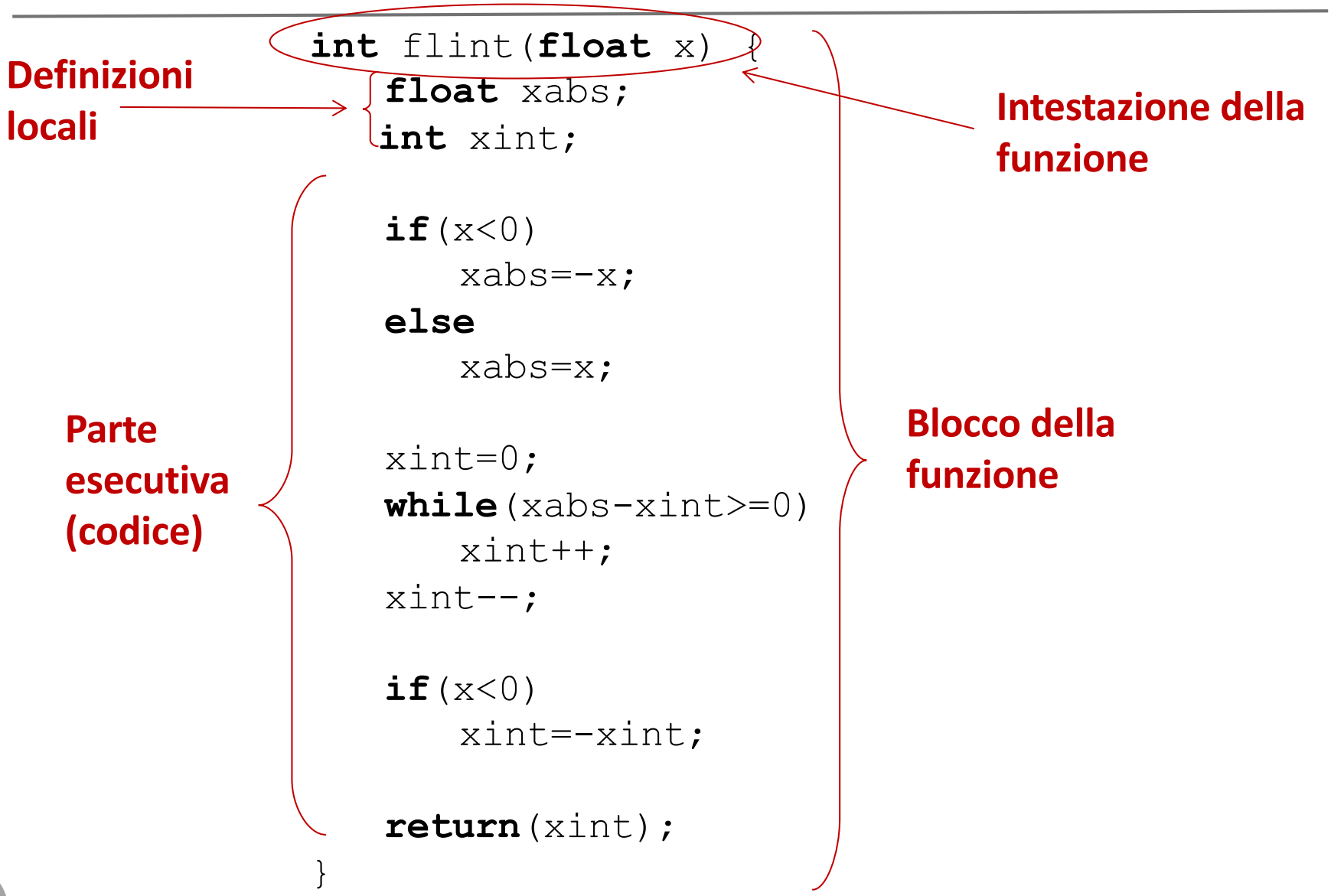
SOTTOPROGRAMMI
FUNZIONI IN C

Carmen De Maio

Struttura di una funzione

- Sono riconoscibili due parti:
  - **l'intestazione**
  - **il blocco**
- L'intestazione della funzione riporta le informazioni principali relative alla funzione: nome, tipo restituito, parametri di ingresso.
- Il blocco è costituito da:
  - una parte dichiarativa (variabili locali)
  - una parte esecutiva (istruzioni)

Struttura di una funzione




```
int flint(float x) {
    float xabs;
    int xint;

    if(x<0)
        xabs=-x;
    else
        xabs=x;

    xint=0;
    while(xabs-xint>=0)
        xint++;
    xint--;

    if(x<0)
        xint=-xint;

    return(xint);
}
```

Nome della funzione

- Il nome identifica univocamente la funzione
- I nomi delle funzioni hanno gli stessi vincoli dei nomi delle variabili. Il nome deve cominciare con una lettera che può essere seguita da una combinazione di lettere, cifre, underscore.

Parte esecutiva

- La parte esecutiva contiene l'insieme di istruzioni che implementa l'operazione che la funzione deve realizzare.
- Le istruzioni lavorano sull'insieme formato dai parametri di ingresso e dalle variabili definite all'interno.
- Le istruzioni possono essere costrutti di qualunque tipo (calcolo e assegnazione, I/O, selezioni, cicli, commenti, linee vuote, chiamate di altre funzioni).
- Al termine c'è una istruzione di **return** il cui scopo è di:
  - terminare l'esecuzione della funzione;
  - restituire il valore tra parentesi come valore della funzione.

```
int flint(float x) {
    float xabs;
    int xint;

    if(x<0)
        xabs=-x;
    else
        xabs=x;

    xint=0;
    while(xabs-xint>=0)
        xint++;
    xint--;

    if(x<0)
        xint=-xint;

    return(xint);
}
```

Funzioni: attivazione

- L'esecuzione delle istruzioni di una funzione è provocata da una particolare istruzione del programma che lo attiva (istruzione di *chiamata*, per cui il programma è anche detto *chiamante*).
- Ciò determina la sospensione dell'esecuzione delle istruzioni del programma chiamante, che riprenderà dopo l'esecuzione dell'ultima istruzione del sottoprogramma (tipicamente, un'istruzione di *ritorno*).

Esempio

(Definizione della funzione)

Nome funzione

Tipo di ritorno

paramentro

```
int flint(float x) {
    ….

    return(xint);
}
```

Il valore restituito coincide con il tipo di ritorno

```
# include <stdio.h>
//non completo!

int main() {
   float x;

   printf("%d ", flint(x) );
   printf("\n");

   return 0;
}
```

Chiamata della funzione

Funzioni: flusso dei dati

- Il passaggio dei dati in ingresso dal programma chiamante alla funzione avviene attraverso una lista di variabili elencate nell'intestazione della funzione, dette **argomenti** o **parametri formali** della funzione. Esse sono destinate ad ospitare i dati di ingresso della funzione.
- Con la istruzione di chiamata, il programma chiamante fornisce alla funzione una lista di **parametri effettivi**, costituiti dai valori su cui la funzione deve *effettivamente* operare.
- La corrispondenza tra parametri effettivi e formali è fissata per ordine.

Parametri Formali E Attuali

- Le funzioni ricevono eventuali parametri dal proprio chiamante
- Nella funzione chiamata:
  - Parametri formali: nomi "interni" dei parametri
- Nella funzione chiamante:
  - Parametri attuali - Valori effettivi (costanti, variabili, espressioni)

```
int flint(float x)
{
    ….
}
```

```
int main() {
….
   printf("%d ", flint(x) );
…
}
```

Parametri Formali E Attuali

- I parametri attuali sono in esatta corrispondenza con i parametri formali
- Dichiarazione e definizione di una funzione devono essere *consistenti*
- Tipi di dato compatibili con i parametri formali:
  - Costanti
  - Variabili
  - Espressioni

```
int func(float a,int b)
{
    ….
}
```

```
int main() {
….
   int x = 4;
   float y = 0.3;
   printf("%d ", func(x, y) );
   printf("%d ", func(y, x) );
…
}
```

Chiamata di una funzione

```
#include <stdio.h>
int main() {
    float a,b;
    int ai,bi;

    printf("a: "); scanf("%f",&a);
    printf("b: "); scanf("%f",&b);

    ai=flint(a);
    bi=flint(b);

    printf( "La parte intera di %f e' %d\n", a, ai);
    printf( "La parte intera di %f e' %d\n", b, bi);

    printf( "Somma delle parti intere: %d\n ", ai+bi);
    printf( "Parte intera della somma: %d\n ", flint(a+b));

    return(0);
}
```

Esecuzione di una funzione

1. Nel programma chiamante, la valutazione di un'espressione attiva la chiamata della funzione;
2. All'atto della chiamata, i parametri effettivi vengono valutati ed assegnati ai rispettivi parametri formali;
3. L'esecuzione del programma chiamante viene sospesa e il controllo viene ceduto alla funzione;
4. Inizia l'esecuzione della funzione: i parametri formali sono inizializzati con i valori dei parametri effettivi;
5. Le istruzioni della funzione sono eseguite;
6. Come ultima istruzione viene eseguito un `return` che fa terminare l'esecuzione della funzione e restituire il controllo al programma chiamante;
7. Continua la valutazione dell'espressione nel programma chiamante sostituendo al nome della funzione il valore restituito.

Esempio

(Definizione della funzione)

```
int flint(float x) {
    ….

    return(xint);
}
```

```
# include <stdio.h>
//non completo!

int main() {
   float x;

   printf("%d ", flint(x) );
   printf("\n");

   return 0;
}
```

Librerie di funzioni

- Alcune funzioni sono già disponibili all'interno di librerie fornite con il compilatore e quindi non richiedono una definizione esplicita da parte dell'utente.
- Es.: **Funzioni della libreria matematica**
- Per utilizzarle occorre:

```
#include<math.h>
```

- Funzioni disponibili:

```
sqrt(x)     radice quadrata
exp(x)      e^x
log(x)      logaritmo naturale
log10(x)  logaritmo base 10
pow(x,y) x^y
sin(x), cos(x), tan(x) trigonometriche (x in radianti)
```

Organizzazione del programma

```c
#include <stdio.h>

int flint(float x) {
      float xabs;
      int xint;

      if(x<0)
            xabs=-x;
      else
            xabs=x;
      xint=0;
      while(xabs-xint>=0)
            xint++;
      xint--;
      if(x<0)
            xint=-xint;
      return(xint);
}

int main() {
      float a,b;
      int ai,bi;

      printf("a: "); scanf("%f",&a);
      printf("b: "); scanf("%f",&b);
      ai=flint(a);
      bi=flint(b);

      printf( "La parte intera di %f e' %d\n", a, ai);
      printf( "La parte intera di %f e' %d\n", b, bi);
      printf( "Somma delle parti intere: %d\n ", ai+bi);
      printf( "Parte intera della somma: %d\n ", flint(a+b));

      return(0);
```

La definizione della funzione `flint` deve precedere il `main`

La funzione **`main`**

- Anche il blocco identificato da **`main`** è una funzione a tutti gli effetti.
- Particolarità di **`main`**:
  - viene chiamata dal Sistema Operativo all'atto dell'esecuzione del programma;
  - il flusso di dati avviene con il S.O., sia per i parametri effettivi in ingresso, sia per il valore restituito da **`return`**

Prototipo di una funzione

- Come per le variabili, anche le funzioni devono essere definite prima di essere usate.
- Nel caso ci siano più funzioni, il **`main`** andrebbe in fondo al file sorgente, rendendo meno leggibile il codice.
- In effetti, per poterle gestire correttamente, il compilatore ha bisogno solo delle informazioni presenti nell'intestazione della funzione.
- E' quindi possibile anticipare al **`main`** solo le intestazioni delle funzioni (prototipi) e inserire dopo il **`main`** le definizioni delle funzioni.
- Il prototipo è formato dall'intestazione della funzione terminato con ';':

```
int flint(float x);
```

Organizzazione del programma con i prototipi

```c
#include <stdio.h>
int flint(float x);

int main() {
      float a,b;
      int ai,bi;

      printf("a: "); scanf("%f",&a);
      printf("b: "); scanf("%f",&b);
      ai=flint(a);
      bi=flint(b);

      printf( "La parte intera di %f e' %d\n", a, ai);
      printf( "La parte intera di %f e' %d\n", b, bi);
      printf( "Somma delle parti intere: %d\n ", ai+bi);
      printf( "Parte intera della somma: %d\n ", flint(a+b));

      return(0);
}

int flint(float x) {
      float xabs;
      int xint;

      if(x<0)
            xabs=-x;
      else
            xabs=x;
      xint=0;
      while(xabs-xint>=0)
            xint++;
      xint--;
      if(x<0)
            xint=-xint;
      return(xint);
}
```

Prototipo della funzione

```c
int flint(float x);
```

Definizione della funzione

Esempio

```
# include <stdio.h>
float terzo_di(float k);
//prototipo della divisione per 3
main() {
    float x= 3.4, y=11.5, z;
    z= terzo_di(x);
    printf("z=%f\n", z);
    z= terzo_di(y);
   printf("z=%f\n", z);
    return 0;
}
//definizione della funzione
float terzo_di(float k) {
 return (k/3);
 }
```

Prima chiamata:
z=terzo_di(3.4)

k=3.4

Seconda chiamata:
z=terzo_di(11.5)

k=11.5

ESERCIZIO

- Scrivere una funzione che calcoli la media dei valori interi in un intervallo dati gli estremi

ESERCIZIO

- Scrivere una funzione che calcoli la media dei valori interi in un intervallo dati gli estremi

```
float media(int a, int b)
{
    int i, somma=0;
    float average;
    for (i=a; i<=b; i++)
        somma +=i;
    average= (float) somma/(b-a+1);
    return  average;
}
```

```c
#include <stdio.h>
float media(int , int);
int main()
{
      int inf, sup;
      float risultato;
      printf("inserisci i limiti dell'intervallo");
      scanf("%d%d", &inf, &sup);
      risultato= media(inf, sup);
      printf("%5.2f", risultato);

      return 0;
}

float media(int a, int b)
{
    int i, somma=0;
    float average;
    for (i=a; i<=b; i++)
        somma +=i;
    average= (float) somma/(b-a+1);
    return  average;
}
```

Procedure

- A volte le operazioni da implementare *non richiedono la produzione di un valore, ma l'esecuzione di un'azione*, come la stampa di valori o la modifica di variabili.
- In questi casi si può utilizzare un tipo diverso di sottoprogramma: la **procedura**.
- La chiamata della procedura avviene mediante una esplicita istruzione di chiamata, costituita dal nome della procedura seguito dalla lista dei parametri effettivi tra ().

Funzioni che restituiscono **`void`**

- In C una procedura viene definita come una funzione che non restituisce valori.
- Questo si realizza tramite il tipo **`void`**.
- Può essere presente l'istruzione **`return`**, che in questo caso ha solo la funzione di terminare l'esecuzione della funzione.

```
void stampa3int(int a,int b,int c) {
    int s;

    printf("Primo valore: %d\n",a);
    printf("Secondo valore: %d\n",b);
    printf("Terzo valore: %d\n",c);

    s=a+b+c;
    printf("Somma: %d\n",s);
    return;
}
```

Chiamata di una procedura

- La chiamata di una procedura avviene con un'istruzione apposita costituita dal nome della procedura seguito dalla lista dei parametri effettivi tra parentesi tonde ().
- L'attivazione viene realizzata nelle stesse modalità viste per la funzione.

```
void stampa3int(int,int,int);

int main() {
    int p,q,r;

    p=2; q=12; r=6;
    stampa3int(p,q,r);

    return(0);
}
```

Passaggio per valore

- La tecnica di corrispondenza tra parametri formali ed effettivi vista finora è detta passaggio per valore (o by value): il valore del parametro effettivo viene copiato nel parametro formale.
- Il parametro formale costituisce quindi una copia locale del parametro effettivo.
- Ogni modifica fatta sul parametro formale non si riflette sul parametro effettivo.

Passaggio per valore

Tutti gli esempi visti finora hanno utilizzato il passaggio per valore

```
void stampa3int(int a,int b,int c) {
     int s;

     printf("Primo valore: %d\n",a);
     printf("Secondo valore: %d\n",b);
     printf("Terzo valore: %d\n",c);

     s=a+b+c;
     printf("Somma: %d\n",s);
     return;
}
```

```
int flint(float x) {
     float xabs;
    int xint;

     if(x<0)
          xabs=-x;
     else
          xabs=x;

     xint=0;
     while(xabs-xint>=0)
          xint++;
     xint--;

     if(x<0)
          xint=-xint;

     return(xint);
}
```

Esempio chiamata per valore

```
#include <stdio.h>
int funct(int);
int main() {
   int x=4, y;
   printf("Prima della chiamata: x=%d\n", x);
   y=funct(x);
   printf("Dopo della chiamata: x=%d\n", x);
   return 0;
}

int funct(int a){
    while(a>0)
       printf("%d\n", a--);
return a;
}
```

```
OUTPUT:

Prima della chiamata: x= 4
4
3
2
1
Dopo della chiamata: x=4
```

Siamo soddisfatti?

Immaginiamo di voler realizzare una funzione che realizzi lo scambio tra due variabili definite nel programma chiamante

```
#include <stdio.h>

void swap(int a, int b) {
   int temp;

   temp = a;
   a = b;
   b = temp;
}

int main()
{
          int x, y;

          printf("Valore di x: "); scanf("%d", &x);
          printf("Valore di y: "); scanf("%d", &y);

          swap(x,y);

          printf("Nuovo valore di x: %d\n", x);
          printf("Nuovo valore di y: %d\n", y);
          return 0;
}
```

Che cosa succede? Perché?

Soluzione

Passaggio per riferimento

- Passa l'**argomento originale**
- Le modifiche fatte all'interno della funzione **hanno effetto** sull'originale

Variabili, registri ed indirizzi

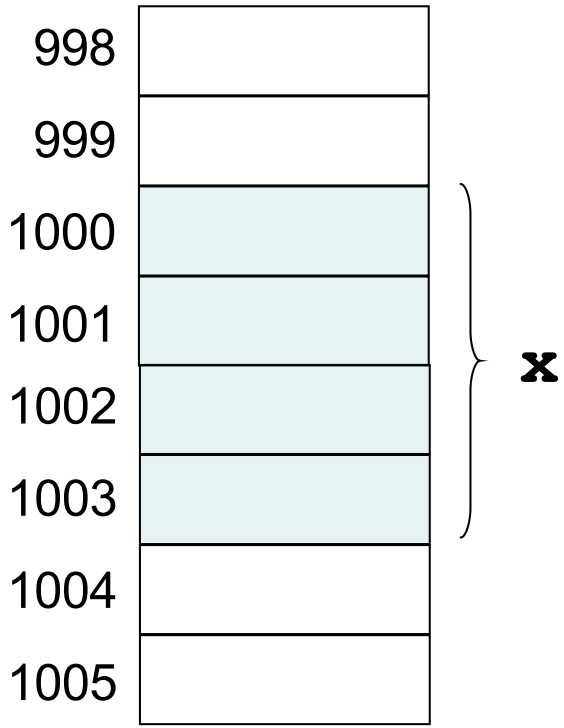
- La definizione di una variabile implica l'allocazione (da parte del compilatore) di registri di memoria. Il numero di registri allocati dipende dal tipo della variabile.
- Alla porzione di memoria allocata si accede tramite l'identificatore della variabile. Questo ci risparmia di preoccuparci in quale particolare locazione la variabile sia realmente allocata.
- È il compilatore a creare e gestire la corrispondenza tra identificatore della variabile e indirizzo della locazione in memoria.

Variabili, registri ed indirizzi

Esempio:

```
int x;
```

Con l'istruzione viene definita una variabile intera **x** che occupa 4 registri da 1 byte a partire dall'indirizzo 1000.



Variabili, registri ed indirizzi

- Il C dà la possibilità di accedere esplicitamente all'indirizzo di una variabile tramite l'operatore **`&`** (operatore di riferimento o di *reference*) prefisso all'identificatore della variabile.

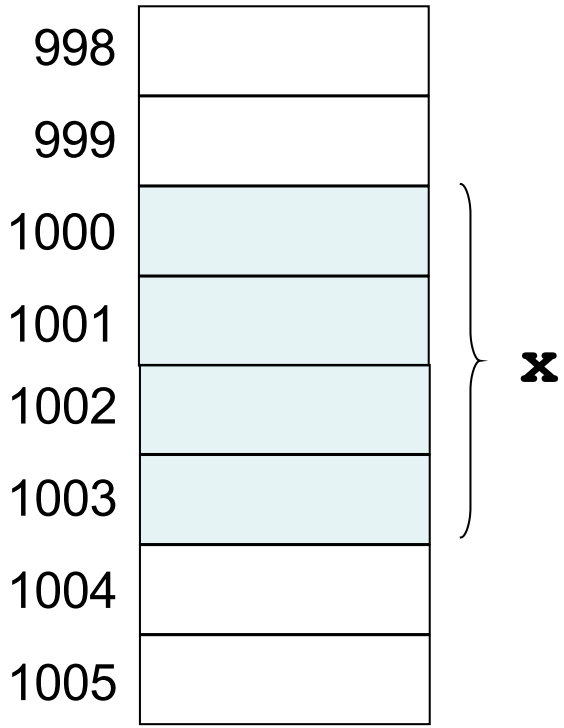
`int x;` variabile **`x`**

`&x` indirizzo della variabile **`x`**

Variabili, registri ed indirizzi

- Esempio:
  `int x;`

- In questo caso `&x` sarà uguale a 1000.



Variabili, registri ed indirizzi

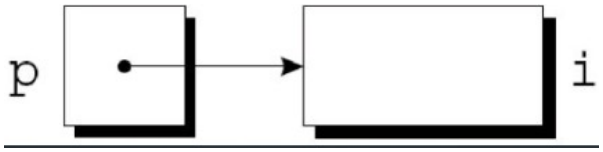
```
#include <stdio.h>
using namespace std;

int main(){
    int x=3;

    printf("Valore di x: %d\n", x);
    printf("Indirizzo di x: %p\n", &x);
    return (0);
}
```

```
Valore di x: 3

Indirizzo di x: 0x7ffe94169bcc
```

indirizzo esadecimale

Puntatori

- Il C permette di definire delle variabili di tipo *puntatore* cui si possono assegnare gli indirizzi di variabili di un particolare tipo.
- Quando memorizziamo l'indirizzo di una variabile **i** nel puntatore **p**, diciamo che **p** "punta a" **i**



- La definizione di tali variabili (dette *puntatori*) richiede la specificazione del tipo "puntato", seguito da un '*****'.
- Es.: definizione di un puntatore a **`int`**

```
int* p;
```

Puntatori

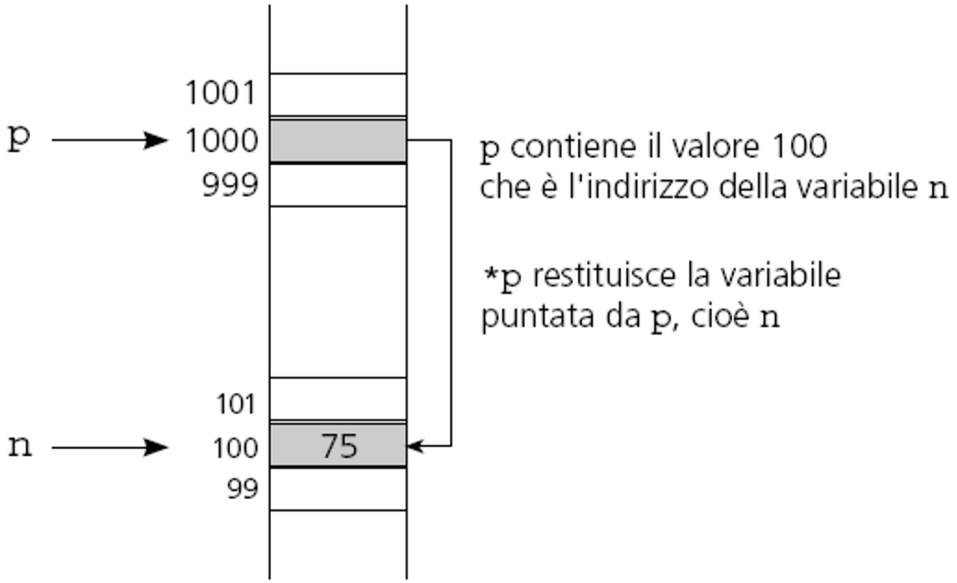
- Tramite il puntatore è possibile accedere alla variabile puntata.
- Con l'operatore '*****' (operatore di indirezione o di *dereference*) prefisso all'identificatore della variabile puntatore è possibile accedere direttamente alla variabile puntata, sia in lettura che in scrittura.
- In questo modo si crea un alias della variabile che può essere modificata tramite il puntatore.

Puntatori

- Di fatto una variabile di tipo puntatore al tipo T contiene l'indirizzo di memoria di una variabile di tipo T.

- Esempio:

```
int n;
int* p;

n = 75;
p = &n;
```

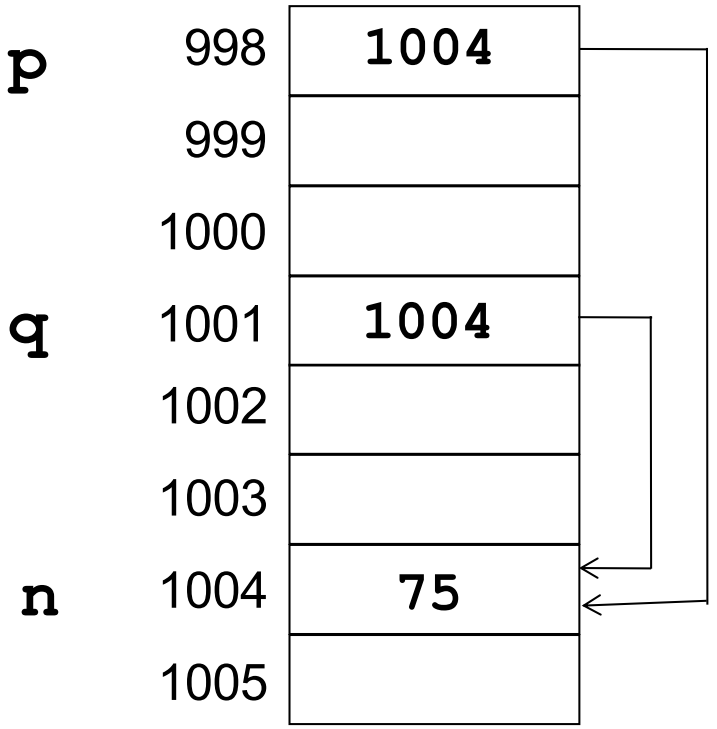


Puntatori

- È possibile fare assegnazioni tra puntatori.

- Esempio:

```
int n=75;
int *p,*q;

 p = &n;
 q = p;
```



- In questo modo due puntatori puntano alla stessa variabile.

Puntatori

```
#include <stdio.h>

int main() {
    int x=3,y=5;
    int *p;

    p = &x;
    *p = 10;
    printf("Valore di x: %d\n", x);

    p = &y;
    *p = 20;
    printf("Valore di y: %d\n", y);

    return 0;
}
```

```
Valore di x: 10
Valore di y: 20
```

Passaggio per riferimento

- Nel passaggio per riferimento, al parametro formale viene assegnato l'indirizzo del parametro effettivo.
- In questo modo, al sottoprogramma è possibile accedere al registro che ospita il parametro effettivo e fare delle modifiche che saranno poi visibili al programma chiamante.
- In altre parole, qualunque modifica effettuata sul parametro formale avrà effetto sul parametro effettivo corrispondente.

Passaggio per riferimento

Il passaggio per riferimento (o by reference) lo si realizza tramite puntatori

```
#include <stdio.h>

void swap(int* a, int* b) {
   int temp;

   temp = *a;
   *a = *b;
   *b = temp;
}

int main()
{
          int x, y;

          printf("Valore di x: "); scanf("%d", &x);
          printf("Valore di y: "); scanf("%d", &y);

          swap(&x,&y);

          printf("Nuovo valore di x: %d\n", x);
          printf("Nuovo valore di y: %d\n", y);
          return 0;
}
```

Passaggio per riferimento

Il passaggio per riferimento (o by reference) lo si realizza tramite puntatori

```
#include <stdio.h>

void incrementa(int* a, int* b, int* c){
    *a = *a+1;
    *b = *b+1;
    *c = *c+1;
}

int main() {
    int x,y,z;

    x=0; y=1; z=2;
    incrementa(&x, &y, &z);
    printf("%d %d %d\n", x, y, z);
    return(0);
}
```

Passaggio per riferimento

- Il passaggio per riferimento fornisce un modo efficace per realizzare una funzione che deve restituire più di un valore

```
void precsucc(int x, int* prec, int* succ){
    *prec=x-1;
    *succ=x+1;
}
```

Esempio SCANF

- Gli argomenti di una chiamata a scanf sono puntatori:

```
int i;
...
scanf("%d", &i);
```

Senza l'operatore **&**, la funzione **scanf** non potrebbe cambiare il valore di **i**

Esercizio

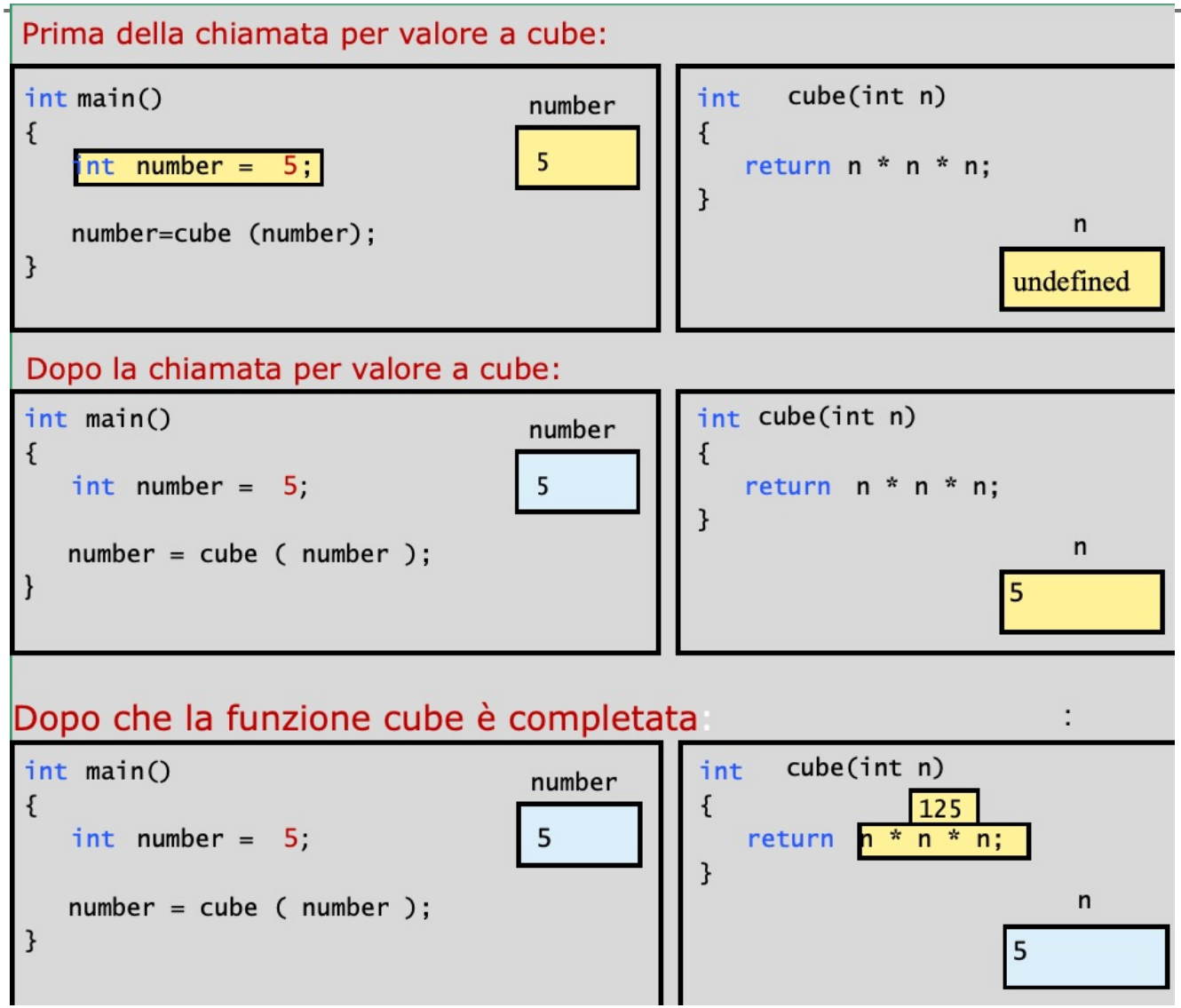
- **Scrivere un programma che calcola il cubo di una variabile mediante chiamata per valore.**

Esercizio

- **Scrivere un programma che calcola il cubo di una variabile mediante chiamata per valore.**

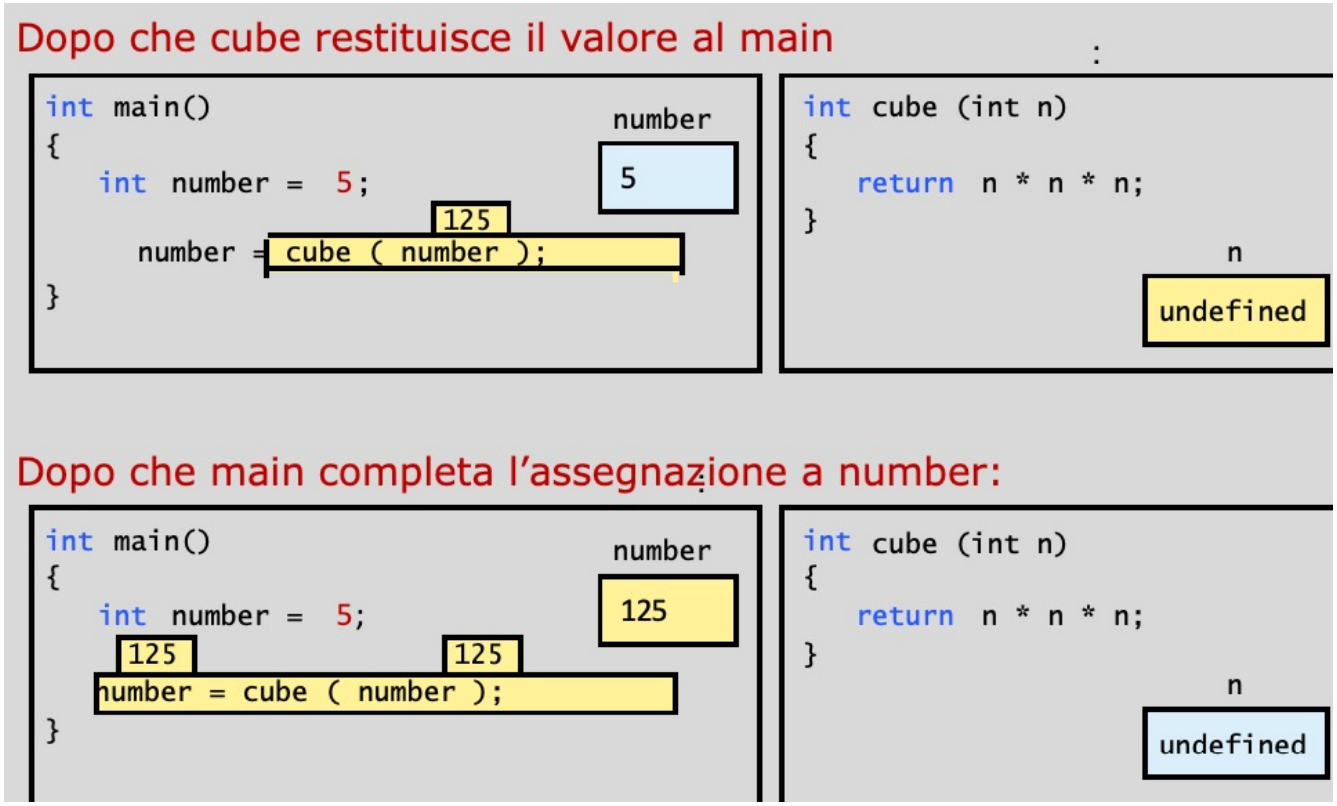
```
#include <stdio.h>
int cube ( int ); /* prototipo */

int main() {
   int number = 5;
   printf( "The original value of number is %d", number );
   number = cube ( number );
   printf( "\nThe new value of number is %d\n", number );
return 0;
}

int cube ( int n ) {
   return n * n * n; /* calcolo del cubo: variabile locale n */
}
```

Passi Di Una Chiamata Per Valore



RV(n) è uguale RV(number)

Passi Di Una Chiamata Per Valore



Esercizio

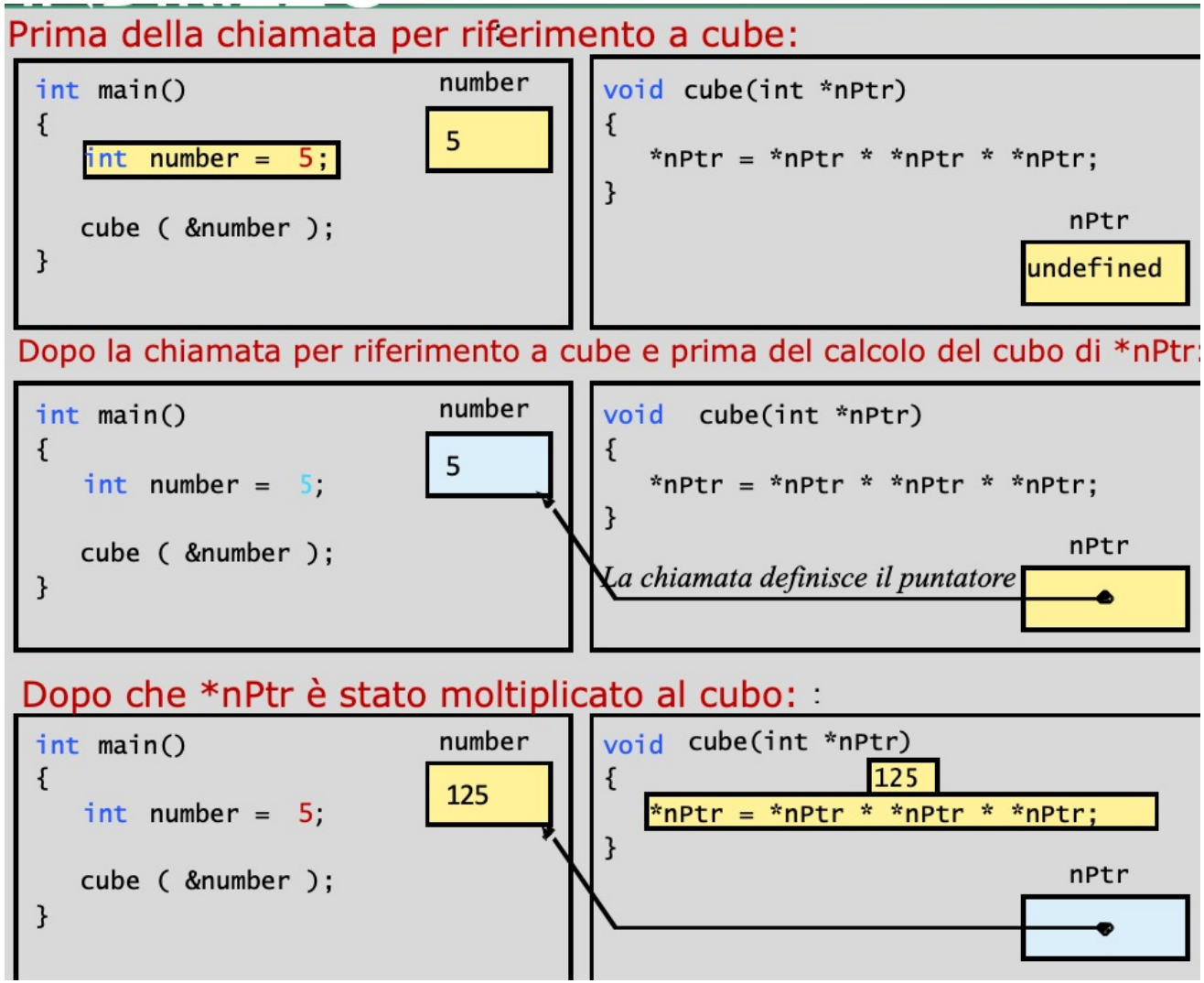
- **Scrivere un programma che calcola il cubo usando una chiamata con un argomento puntatore.**

Esercizio

- **Scrivere un programma che calcola il cubo usando una chiamata con un argomento puntatore.**

```
#include <stdio.h>
void cube ( int * ); /* prototipo */

int main() {
   int number = 5;
   printf( "The original value of number is %d", number );
   cube ( &number );
   printf( "\nThe new value of number is %d\n", number );
   return 0;
}

void cube ( int *nPtr ) {
   *nPtr = *nPtr * *nPtr * *nPtr; /* il cubo di number n main */
}
```

Passi di una chiamata Per Indirizzo



RV(nPtr) è uguale LV(number)

Esercizio

- Robin Hood è un ladro gentiluomo: ruba ai ricchi per dare ai poveri, ma siccome è gentiluomo non vuole far diventare poveri i ricchi!
- Due persone hanno una quantità di denaro x e y, richiamando la funzione robinHood, equilibrare la quantità di denaro tra i due.

Esercizio

```
#include <stdio.h>
#include <math.h>


void robinHood(float* d1,float* d2)
{
   *d1=(*d1+*d2)/2;
   *d2=*d1;
}

int main()
{
   float x,y;
  printf("inserisci due numeri ");
   scanf("%f %f", &x, &y);
   robinHood(&x,&y);
   printf("x= %f , y= %f", x,y);
}
```