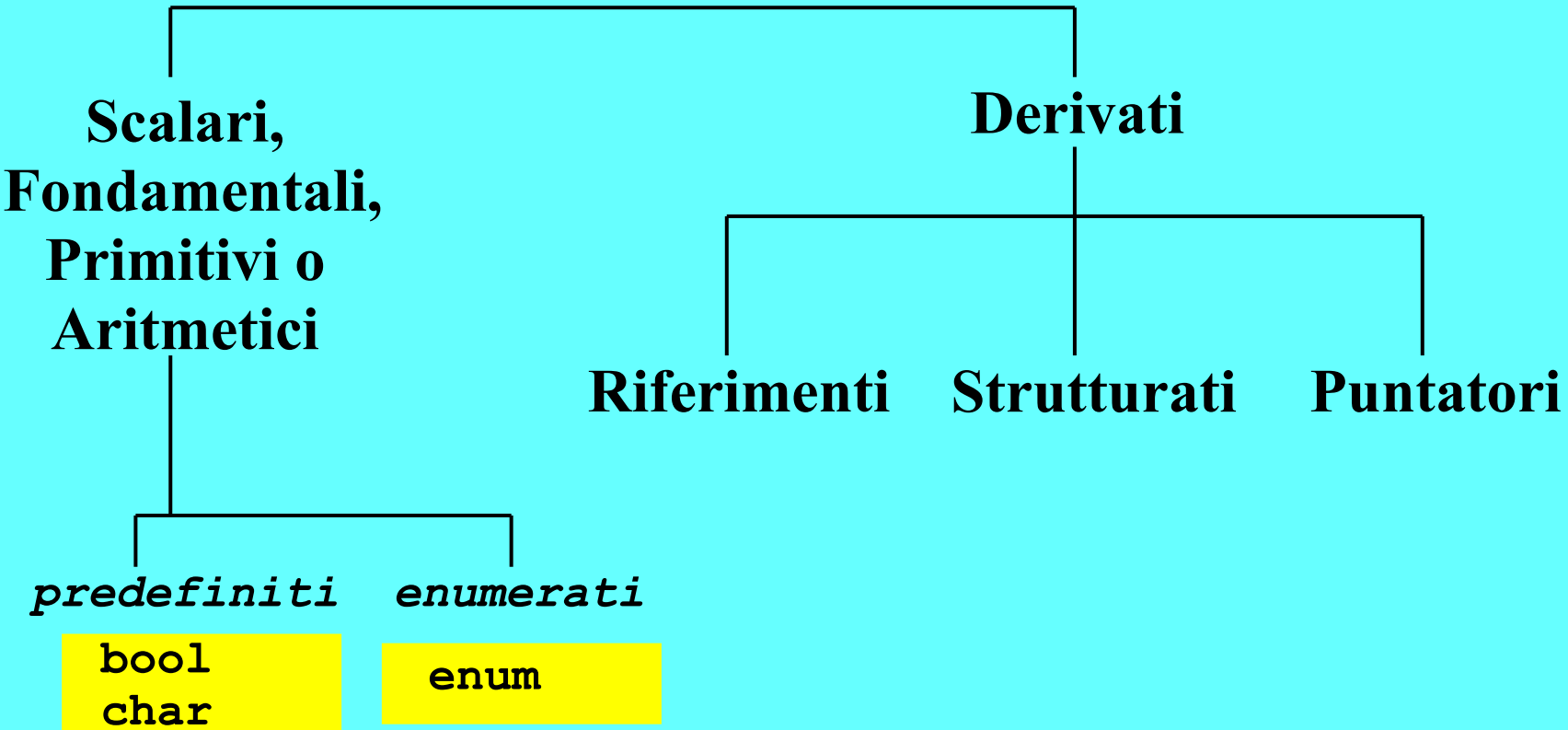
Riferimenti e funzioni

(Passaggio dei parametri *per riferimento*)

Tipi di dato


Scalari, Fondamentali, Primitivi o Aritmetici
Derivati
Riferimenti
Strutturati
Puntatori
predefiniti
enumerati
bool
char
int
float
double
enum

Riferimento

- Identificatore associato all'indirizzo di un oggetto
- Quando si dichiara un oggetto (costante, variabile, funzione, ...) gli si assegna un riferimento di *default*
  - Es.:
    ```
    int a ; // l'dentificatore a è il riferimento di
                default alla variabile di nome a
    ```
- Oltre a quello di default, in C++ (non in C) è possibile definire ulteriori riferimenti ad un oggetto
  - Rappresenteranno dei sinonimi (alias)

Riferimenti come variabili locali o globali

- <definizione_riferimento> ::=
  <tipo_riferimento> identificatore = <nome_oggetto>
  <tipo_riferimento> ::=
  <tipo_oggetto> **&**

- Es:

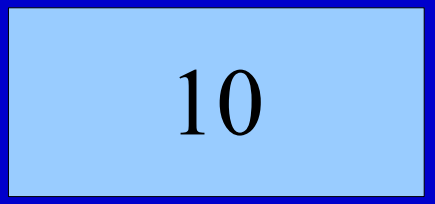
```
int i = 10 ;
int & rif_ad_i = i ;
rif_ad_i = 5 ;

cout<<i<<" "<<rif_ad_i<<endl ;
```

- **Cosa stampa ?**

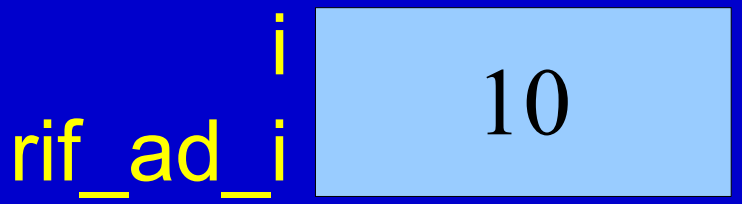
Riferimenti come variabili locali o globali

- Una volta definito ed inizializzato, un riferimento si riferisce per sempre allo stesso oggetto
- Nell'esempio precedente:

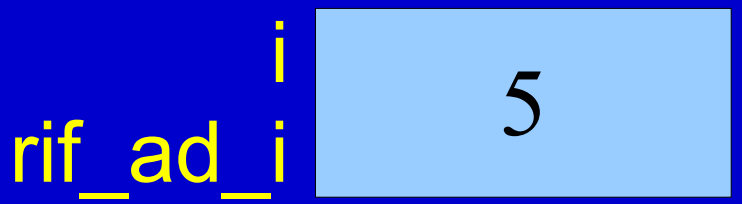
```
int i = 10 ;
```




```
int & rif_ad_i = i ;
```




```
rif_ad_i = 5 ;
```



Passaggio dei parametri

- Per “passaggio dei parametri” si intende l’associazione fra parametri attuali e parametri formali che avviene al momento della chiamata di una funzione

- ***Il meccanismo di base adottato in C++ (l'unico possibile in C), si chiama PASSAGGIO DI PARAMETRI PER VALORE***

- Non è il solo meccanismo possibile

- **Il C++, come altri linguaggi fornisce anche il passaggio per riferimento**

Passaggio dei parametri per valore

Le locazioni di memoria corrispondenti ai parametri formali:

- Sono allocate al momento della chiamata della funzione
- Sono inizializzate con i valori dei corrispondenti parametri attuali trasmessi dalla funzione chiamante
- Vivono per tutto il tempo in cui la funzione è in esecuzione
- Sono deallocate quando la funzione termina

Passaggio parametri per valore (*cont.*)

QUINDI:

- La funzione chiamata riceve copia dei valori dei parametri attuali passati dalla funzione chiamante
- Tali copie sono *sue copie private che servono solo per inizializzare i parametri formali*
- Ogni modifica ai parametri formali è *strettamente locale* alla funzione

- **I parametri attuali della funzione chiamante non saranno mai modificati!**

Perché il C adotta il passaggio per valore?

- E' sicuro: le variabili del chiamante e del chiamato sono *completamente disaccoppiate*
- Consente di ragionare per componenti e servizi: *la struttura interna dei singoli componenti è* ***irrilevante***

  (la funzione può anche modificare il contenuto dei parametri formali, inizializzati ai valori dei parametri attuali, senza che ciò abbia alcun impatto sui valori dei parametri attuali nell'ambiente del chiamante)

Limite del passaggio per valore

- **Il passaggio per valore ha una SEMANTICA PER COPIA**
  - ***Impedisce a priori* di scrivere funzioni che abbiano come scopo quello di *modificare* i dati passati dall'ambiente chiamante**
  - (*collegato al precedente*) **Impedisce di restituire più di un risultato all'ambiente chiamante**
  - **Può essere costoso per oggetti di grosse dimensioni**

Esempio (*Programma scambia*)

```
void scambia(int A, int B)
  { int t;
    t = A;   A = B;   B = t;
  }

main()
{
  int  x = 12, y = 27;
  cout<<"x="<<x<<" y="<<y<<endl ;
  scambia(x, y);
  cout<<"x="<<x<<" y="<<y<<endl ;
}
```

Stampa voluta	***Stampa vera***
`x=12  y=27`	`x=12  y=27`
`x=27  y=12`	`x=12  y=27`

MOTIVO: Semantica del **passaggio di parametri per valore.**

La funzione **scambia** ha scambiato A e B **al suo interno, ma questa modifica non si è propagata** ai parametri attuali

Passaggio dei parametri per riferimento

- La versione corretta del programma C++ che realizza la funzione **scambia()** necessita di un **passaggio di parametri per riferimento**

- In pratica, per superare i limiti della semantica per copia, occorre consentire alla funzione di **far riferimento alle variabili dell'ambiente del chiamante**

Principî del passaggio per riferimento

- Una funzione deve poter dichiarare, nella sua intestazione, che un **parametro costituisce un riferimento**
- In tal caso:
  - il parametro formale indicato come “riferimento”non è più una variabile locale inizializzata al valore del parametro attuale, ma è un riferimento alla variabile originale (*parametro attuale*) nell’ambiente della funzione chiamante
  - quindi, ogni modifica fatta al parametro formale, in realtà viene effettuata sul parametro attuale della funzione chiamante (le modifiche fatte dall’ambiente chiamato si propagano all’ambiente chiamante)

Passaggio parametri per riferimento in C

Il passaggio per riferimento è disponibile in molti linguaggi di alto livello (C++, Pascal), ma non in tutti.

In particolare, NON è disponibile in C, e quindi in C deve essere simulato tramite l'uso di puntatori

Riferimenti come parametri formali

- <definizione_riferimento> ::=
  <tipo_riferimento> identificatore
  <tipo_riferimento> ::=
  <tipo_oggetto> **&**

- **Esempio:**
  Scrivere una funzione `raddoppia` che prenda in ingresso (parametro formale) un oggetto di tipo intero e ne raddoppi il valore

Riferimenti come parametri formali

```
#include <iostream>
using namespace std ;

void raddoppia(int &a)
{
   a *= 2 ;
}

int main()
{
   int b ;
   cin>>b ;
   raddoppia(b) ;
   cout<<b<<endl ;
}
```

Esercizio ...

Restituzione di più di un risultato ...

- Definire una funzione che prenda in ingresso (parametro formale) un oggetto di tipo carattere e, se si tratta di un carattere maiuscolo, lo trasformi nel corrispondente carattere minuscolo. La funzione deve ritornare vero se la conversione è avvenuta, falso altrimenti.
- Tipo dei parametri in ingresso?
- Tipo del valore di ritorno?

Finalmente la funzione “scambia”

```
void scambia(int &A, int &B)
  { int t;
    t = A; A = B; B = t;
  }

main()
{
  int  x = 12, y = 27;
  cout<<“x=”<<x<<” y=”<<y<<endl ;
  scambia(x, y);
  cout<<“x=”<<x<<” y=”<<y<<endl ;
}
```

Stampa

`x=12  y=27`

`x=27  y=12`

E' cambiato qualcosa nel **main** rispetto alla precedente versione ???

Problemi passaggio per riferimento

- Dalla sola chiamata di una funzione non si può distinguere tra un passaggio per valore ed uno riferimento
- Perché i passaggi per riferimento possono essere pericolosi?

Effetti collaterali

- Effetto collaterale
  - Di una parte A di un programma su un'altra parte B
  - La parte A modifica variabili visibili nella parte B
  - Es: una funzione modifica variabili visibili in un'altra
- Può essere utilizzato come metodo di interazione tra parti di un programma
- Se non previsto o voluto può portare ad errori difficili da scoprire

Attenzione: Effetti collaterali (*veri*)

```
float  a, b;

float R (float& x)
 {
  x=x+x;
  return x;
 }

 main()
  {
   a=1.0;
   b=2*R(a); cout<<b<<endl;
   a=1.0;
   b=R(a)+R(a); cout<<b<<endl;
  }
```

Stampa
4
6

Effetti collaterali (*apparenti*)

```
float  a, b;

float R1 (float &x)
 {
  return x+x;
 }

 main()
  {
   a=1.0;
   b=2*R1(a); cout<<b<<endl;
   a=1.0;
   b=R1(a)+R1(a); cout<<b<<endl;
  }
```

Nel caso precedente (R) si ha un effetto collaterale indesiderato.

In questo caso (R1) non si ha effetto collaterale → si stampa due volte 4

MOTIVO: L'effetto collaterale si può avere solo se il parametro attuale è passato per riferimento e all'interno della funzione il parametro formale subisce delle modifiche.

Effetti collaterali

- I riferimenti forniscono quindi un meccanismo per avere effetti collaterali
- E' l'unico ???

Riassunto interazioni tra parti di un programma

- Valore di ritorno delle funzioni
- Effetti collaterali
  - Variabili globali
  - Riferimenti
- In generale è meglio passare valori alle funzioni che sfruttare effetti collaterali

Riferimento come tipo di ritorno

- **Una funzione può avere un riferimento come tipo di ritorno**
- **Questo permette di "legare" due variabili mediante una funzione**

Riferimento come tipo di ritorno

```
int & fun(int &a)
{
      return a ;
}

main()
{
      int c = 10 ;
      int &b = fun(c) ;
      b++ ;
      cout<<b<<" "<<c<<endl ;
}
```

Cosa stampa?

Riferimento come tipo di ritorno

```
int a = 10 ;
int & fun()
{
      return a ;
}

main()
{
      int &b = fun() ;
      b++ ;
      cout<<b<<" "<<a<<endl ;
}
```

Cosa stampa?

Attenzione !!!

```
int & fun()
{
      int d = 3 ;
      return d ;
}

main()
{
      int &b = fun() ;
      b++ ;
      cout<<b<<endl ;
}
```

E' corretto?

Qual è il tempo di vita della variabile d?

Attenzione !!!

- Ritornare un riferimento ad una variabile locale è un errore logico
- E' anche un errore di gestione della memoria
  - Se poi si assegnano valori a tali riferimenti si corrompe la memoria