Riferimenti e funzioni

(Passaggio dei parametri *per riferimento*)

Esercizio

- **Scrivere una funzione che scambi due valori A e B (di tipo int)**

Idea e Algoritmo

- Effettuare una “triangolazione” fra A, B e una variabile ausiliaria T

- Salvare in T il valore di A, poi assegnare ad A il valore di B, quindi assegnare a B il valore di T (cioè il vecchio valore di A)

Programma

```
void scambia(int A, int B)
  { int 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 la propria copia dei valori di A e B **ma questa modifica non si è propagata** ai parametri attuali

Passaggio dei parametri per riferimento

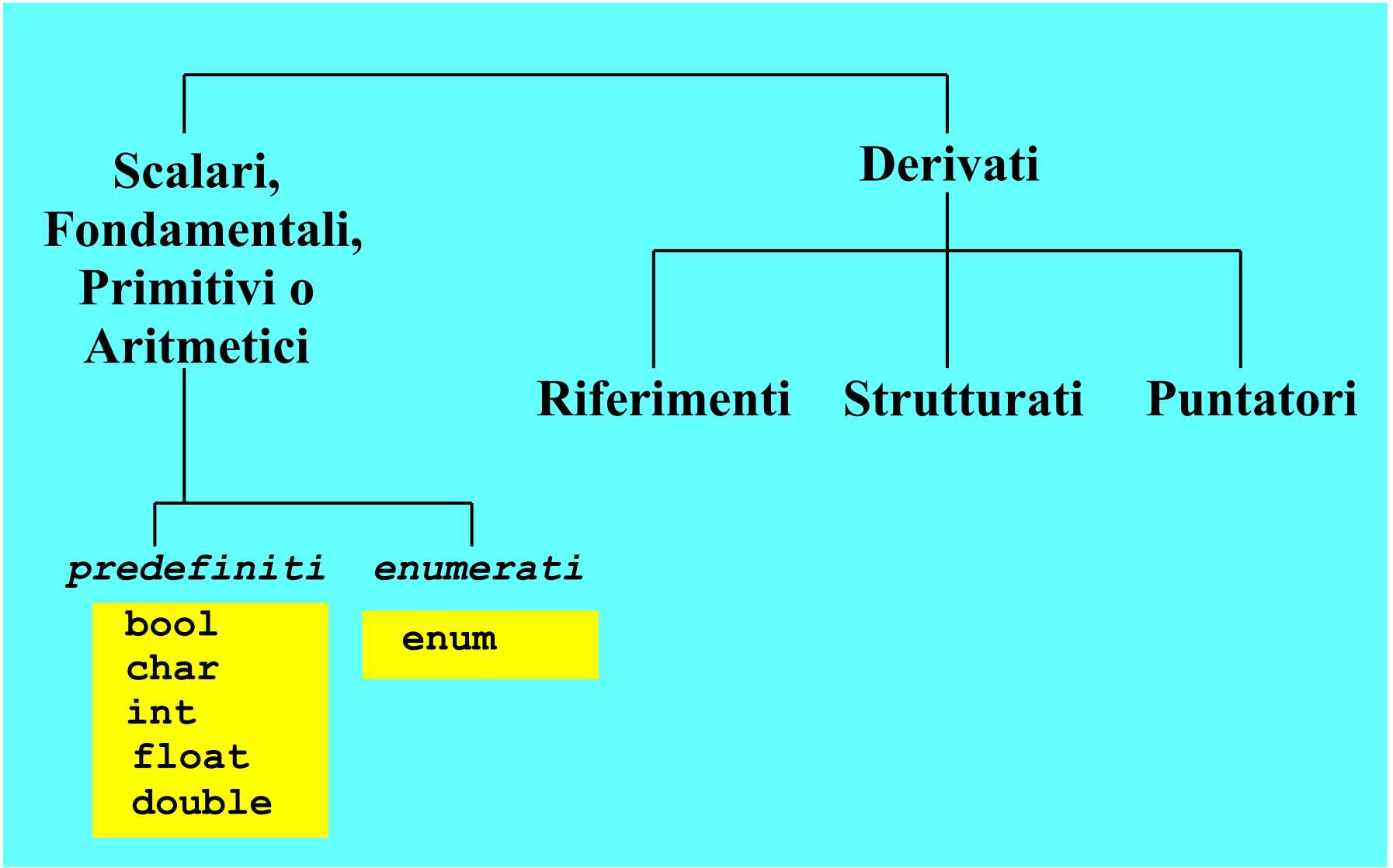
Per superare i limiti della semantica per copia, occorre consentire alla funzione di far riferimento alle variabili effettive del chiamante

IL CONCETTO DI RIFERIMENTO

- Una funzione deve poter dichiarare, nella sua intestazione, che un parametro costituisce un riferimento. In tal caso:
  - il parametro formale non è più una variabile locale inizializzata al valore del parametro attuale, ma è un riferimento alla variabile originale (passata come *parametro attuale*) nell'ambiente della funzione chiamante
  - quindi, ogni modifica fatta al parametro formale in realtà viene effettuata sulla variabile della funzione chiamante passata come parametro attuale (le modifiche si propagano)
- **Il passaggio per riferimento è disponibile in molti linguaggi (Pascal, C++), ma NON è disponibile in C, nel quale deve essere simulato tramite puntatori**

- Saremo presto in grado di realizzare la versione corretta del programma C/C++ che realizza la funzione **scambia()**, non appena introdotto il tipo derivato *riferimento*

Tipi di dato


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

Riferimento

- Un riferimento è un identificatore associato all'indirizzo di un oggetto
- Quando si dichiara un oggetto (costante, variabile, funzione, ...) gli si assegna un riferimento di *default*, ossia quello che finora abbiamo chiamato *nome* dell'oggetto
  - 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:

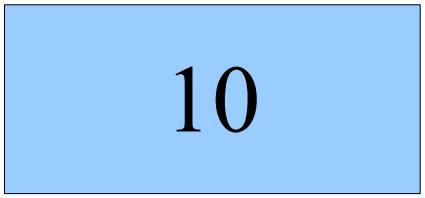
```
int i = 10 ;
int & rif_ad_i = i ;
rif_ad_i = 5 ;

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

- **Cosa stampa ?**

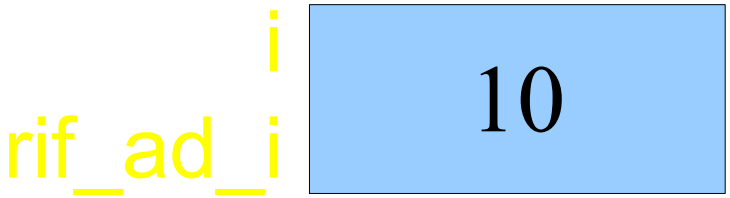
Riferimenti come variabili locali o globali

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

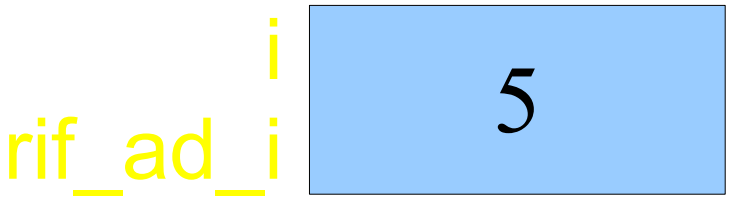
```
int i = 10 ;
```




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




```
rif_ad_i = 5 ;
```



Passaggio dei parametri per 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 visibili nella funzione 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 passata dalla funzione chiamante come parametro attuale
  - 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)

Riferimenti come parametri formali

- <definizione_param_formale_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 ...

- Dalle slide di esercitazione:
  - *funz_moltiplica.cc*

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** (ossia nell'invocazione) 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 per riferimento
- Perché i passaggi per riferimento possono essere pericolosi?

Effetti collaterali

- Si ha un **effetto collaterale**
  - Di una parte A di un programma su un'altra parte B
  - se 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

```
float  a, b;

float R (float& x)
 {
  x *= 2;
  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

- 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 ed utilizzarne il valore di ritorno piuttosto che sfruttare effetti collaterali

Implementazione riferimenti 1/3

- A basso livello il compilatore realizza un riferimento mediante una variabile nascosta in cui memorizza l'indirizzo dell'oggetto di cui il riferimento è un sinonimo

- Ad esempio, dato
  int &a = b ;
  Il compilatore memorizza in una variabile nascosta l'indirizzo di *b*, ed usa tale indirizzo tutte le volte che si deve accedere a *b* attraverso il riferimento *a*

Implementazione riferimenti 2/3

- Pertanto, se si passa un oggetto per riferimento, di fatto si passa solo l'indirizzo dell'oggetto

- Come vedremo, mediante i tipi derivati si possono definire oggetti molto grandi (ossia che occupano molte celle di memoria)
  - Il passaggio per valore di un oggetto diviene tanto più costoso, sia in termini di tempo che di occupazione di memoria, quanto più grande è l'oggetto da passare (si deve copiare l'intero oggetto nel parametro formale)

Implementazione riferimenti 3/3

- Al contrario, il passaggio di un oggetto per riferimento ha sempre lo stesso costo, indipendentemente dalle dimensioni dell'oggetto passato
- E' quindi molto più conveniente passare un oggetto molto grande per riferimento rispetto a passarlo per valore
- Rispetto al passaggio per valore c'è però il problema degli effetti collaterali!
- Come risolverlo ?

Riferimenti ad oggetti costanti

- Se si aggiunge il qualificatore *const* nella definizione di un riferimento, non si potrà mai modificare l'oggetto acceduto attraverso quel riferimento

- Es.:

```
int fun(const int &a)
{
    a = 3 ; // ILLEGALE, non compila
}
```

- Pertanto, mediante il passaggio per riferimento ad oggetto costante possiamo realizzare il passaggio per valore a costo costante

Direzione parametri 1/3

- Un ultimo aspetto importante nell'interazione tra parti di un programma è la *direzione* dei parametri delle funzioni
  - Parametri di ingresso: valori o oggetti su cui lavorerà la funzione
    - Questo tipo di passaggio si può implementare ad esempio con il passaggio per valore o con il passaggio per riferimento ad oggetto costante

Direzione parametri 2/3

- Parametri di uscita: oggetti che non saranno mai usati in lettura, ma solo per memorizzare dei valori di ritorno
  - Permettono di fatto di realizzare funzioni con un vettore di valori di ritorno
    - Mediante l'istruzione *return* si comunica il valore di ritorno della funzione in senso stretto, mentre negli altri parametri di uscita si memorizzano gli altri eventuali valori di ritorno

Direzione parametri 3/3

- Parametri di ingresso/uscita: oggetti usati sia come parametri di ingresso che come parametri di uscita

Restituzione di più di un risultato ...

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

Soluzione

```
/*
 * Funzione che dato un carattere in ingresso, se si tratta di un
 * carattere  minuscolo lo trasforma in maiuscolo e ritorna
 * vero; in caso contrario lo lascia invariato e ritorna falso
*/
bool maiuscolo(char &c)
{
    if (c<'a' || c>'z')
                return false ;
     c = c – 'a' + 'A' ;
    return true ;
 }
```

Riferimento come tipo di ritorno

- **Una funzione può avere un riferimento anche 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**
  - **Il compilatore può non segnalare l'errore**
  - **Se poi si assegnano valori tramite tali riferimenti si può corrompere la memoria del programma**

Passaggio parametri per riferimento in C

Come già detto, 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