Lezione 11

Tipo derivato riferimento
Passaggio per riferimento
Parametri di ingresso e/o uscita

Esercizio

- Scrivere una funzione tramite la quale sia possibile scambiare il valore di due variabili locali al blocco dall'interno del quale viene invocata
- Ad esempio
  - se invocata da dentro il **main**
  - e supponendo che il **main** contenga due variabili locali A e B di tipo **int**,

  scambi i valori memorizzati in tali due variabili

Possibile idea ed algoritmo

- Utilizzare 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  A = 12, B = 27;
  cout<<"A="<<A<<" B="<<B<<endl ;
  scambia(A, B);
  cout<<"A="<<A<<" B="<<B<<endl ;
}
```

- Cosa vorremmo che stampasse?

Stampa desiderata

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

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

Stampa voluta

```
A=12   B=27

A=27   B=12
```

- Cosa stampa?

Stampa effettiva

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

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

Stampa voluta

```
A=12   B=27

A=27   B=12
```

Stampa vera

```
A=12   B=27

A=12   B=27
```

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

La funzione **scambia** ha scambiato **la propria copia** dei valori di A e B, quindi **questa modifica non si è propagata** ai parametri attuali

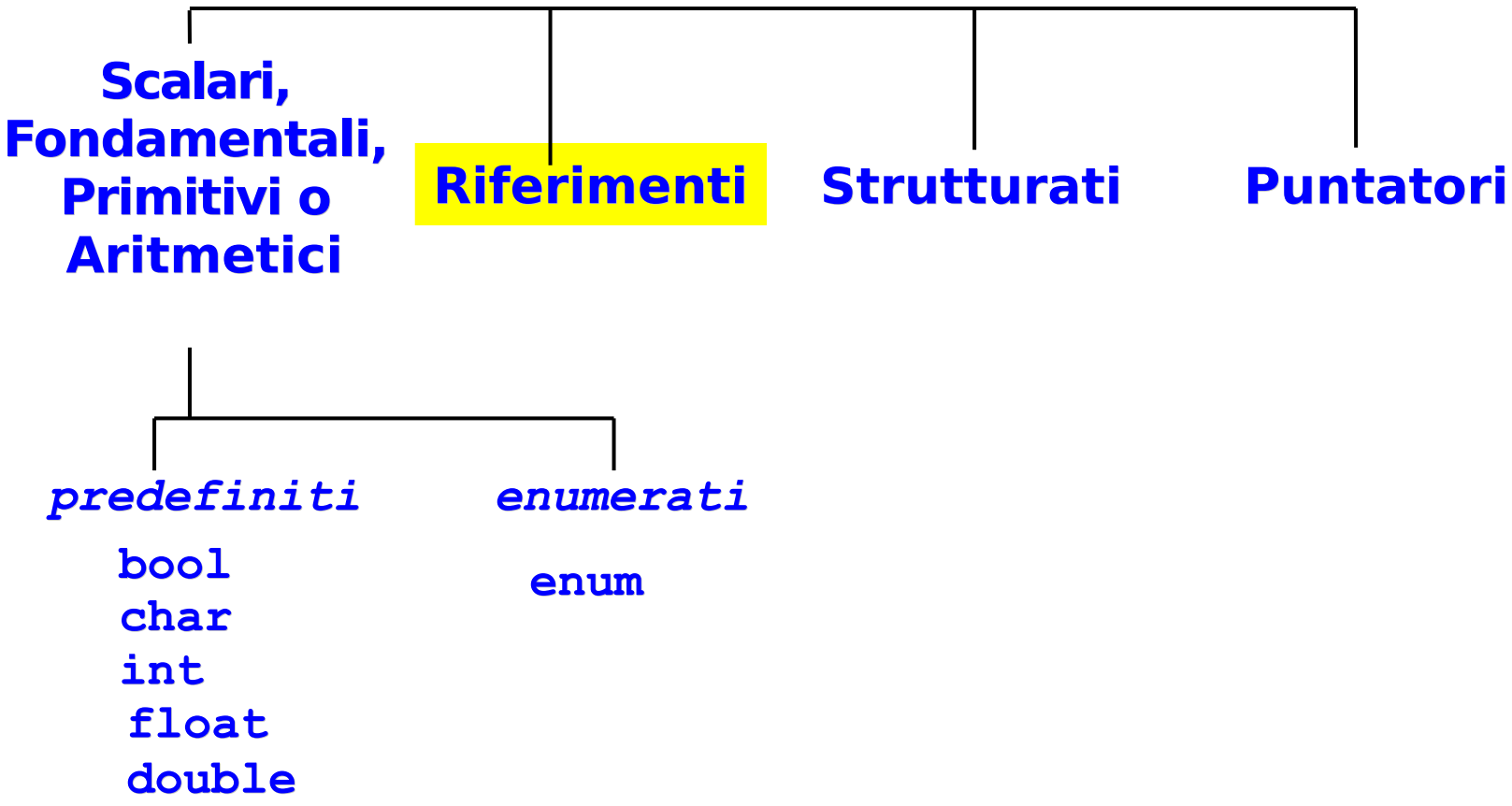
Passaggio per riferimento 1/2

- Per superare i limiti della semantica per copia, occorre consentire alla funzione di far riferimento alle variabili di chi la invoca
- Serve il concetto di passaggio per **RIFERIMENTO**, a cui diamo il seguente significato
  - Se una funzione dichiara, nella sua intestazione, che un parametro costituisce un riferimento, allora
    - il parametro formale non è più una variabile locale dello stesso tipo del parametro attuale ed inizializzata al valore di tale parametro attuale,
      - ma è un riferimento alla variabile passata come parametro attuale nella 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)

Passaggio per riferimento 2/2

- Il passaggio per riferimento è disponibile in molti linguaggi (Pascal, C++)
  - Ma NON è disponibile in C, nel quale deve essere simulato tramite puntatori
  - Vedremo qualcosa sui puntatori nelle prossime lezioni
- Per arrivare al passaggio per riferimento bisogna prima introdurre il **tipo derivato riferimento**
  - Col passaggio per riferimento potremo poi realizzare, ad esempio, la versione corretta della funzione **`scambia()`**

Tipi di dato


Scalari, Fondamentali, Primitivi o Aritmetici
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 già** un riferimento di *default*
  - Tale riferimento è per la precisione quello che finora abbiamo chiamato *nome dell'oggetto*
  - Esempio:

```
int a ; // l'identificatore a è il riferimento
        // di default alla variabile appena
        // definita
```

- Oltre a quello di default, in C++ (non in C) è possibile definire ulteriori riferimenti ad uno stesso oggetto
  - Rappresenteranno dei sinonimi (*alias*)

Riferimenti in C++

- **Sintassi:**
  <definizione_riferimento> ::=
  <tipo_riferimento> *<identificatore>* **=** *<nome_oggetto>* **;**

  <tipo_riferimento> ::= *<tipo_oggetto>* **&**

- Esempio:

```
int i = 10 ;
int & rif_ad_i = i ; // definizione riferimento
rif_ad_i = 5 ;

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

- Cosa stampa ?

Risposta

- Stampa
  `5 5`
- Il riferimento **`rif_ad_i`** è solo un nome alternativo per **`i`**

```
int i = 10 ;
```

i 10

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

i 10
rif_ad_i

```
rif_ad_i = 5 ;
```

i 5
rif_ad_i

Inizializzazione riferimento

- L'inizializzazione di un riferimento all'atto della sua definizione è obbligatoria
- Una volta definito ed inizializzato, un riferimento si riferisce **per sempre allo stesso oggetto**

Passaggio dei parametri per riferimento

Passaggio per riferimento 1/2

- Per passare un parametro attuale per riferimento basta definire il corrispondente parametro formale come un oggetto di tipo riferimento
- **Sintassi:**
  <definizione_param_formale_riferimento> ::=
  [const] *<tipo_riferimento>* *<identificatore>*

  <tipo_riferimento> ::= *<tipo_oggetto>* **&**
- Se si aggiunge il qualificatore **const**, non si può modificare il valore dell'oggetto a cui si riferisce il parametro formale (ci torniamo fra qualche slide)
- Esempio:

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

Passaggio per riferimento 2/2

- La sintassi del **passaggio** di un parametro attuale **per riferimento** all'atto di una chiamata di funzione è **identica a** quella del **passaggio per valore**
- Esempio:

```
fun(int &a) { a++ ; }
main() {
        int i = 0 ;
        fun(i) ;
        cout<<i<<endl ;
}
```

- Cosa stampa ?

Risposta

- Stampa

  1

Riepilogo

- Un parametro formale di tipo riferimento ha esattamente le caratteristiche che ci servono per realizzare il passaggio per riferimento precedentemente introdotto
  - tale parametro formale **non è più una variabile/costante locale** dello stesso tipo del parametro attuale ed inizializzata al valore di tale parametro attuale,
    - ma è un riferimento all'oggetto passato come parametro attuale nella funzione chiamante
  - quindi, ogni modifica fatta al parametro formale in realtà viene effettuata sull'oggetto della funzione chiamante passata come parametro attuale (le modifiche si propagano)
    - affinché si possano legalmente inserire delle istruzioni che modificano un parametro attuale attraverso un parametro formale di tipo riferimento,
      - l'oggetto passato come parametro attuale deve ovviamente essere una variabile
      - non si deve utilizzare il qualificatore **`const`** nella definizione del parametro formale

Esercizio

- Scrivere
  - una funzione **`raddoppia`**, con tipo di ritorno **`void`**, che prenda in ingresso (parametro formale) un oggetto di tipo intero e ne raddoppi il valore
  - una funzione **`main`** che
    - definisca una variabile di tipo **`int`** e ne legga il valore da *stdin*
    - poi invochi la funzione raddoppia per raddoppiare il valore della variabile
    - infine stampi il valore della variabile

Soluzione

```
#include <iostream>
using namespace std ;
void raddoppia(int &) ;

int main()
{
      int b ;
      cin>>b ;
      raddoppia(b) ;
      cout<<b<<endl ;
}
void raddoppia(int &a)
{
      a *= 2 ;
}
```

Esercizio

- Scrivere l'intestazione corretta della funzione scambia che abbiamo provato a scrivere all'inizio di questa lezione

Funzione scambia corretta

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

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

Stampa

```
x=12   y=27

x=27   y=12
```

E' cambiato qualcosa nel **main** (ossia nell'invocazione) rispetto alla precedente versione del programma ???

Esercizio per casa

- Svolgere l'esercizio *funz_moltiplica.cc* dell'ottava esercitazione

Problema

- 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?
- Per la stessa ragione per cui sono utilizzati, come discusso nelle prossime slide ...

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
  - Esempio: una funzione che modifica variabili visibili in un'altra
- Gli effetti collaterali sono uno dei metodi di interazione tra parti di un programma
  - Però se non previsti o voluti possono portare ad errori difficili da scoprire

Esempio

```
int  a, b;
int R (int& x)
{
   x *= 2;
   return x;
}

main()
{
   a=1;
   b=2*R(a); cout<<b<<endl;
   a=1;
   b=R(a)+R(a); cout<<b<<endl;
}
```

Stampa

4

6

Domanda

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

Risposta

- No, ci sono, come sappiamo, anche le variabili globali!

Scelta modalità di interazione

- Riassumendo, parti diverse di un programma possono interagire tramite i seguenti meccanismi
  - Passaggio per valore e ritorno di valori da parte delle funzioni
  - Effetti collaterali
    - Variabili globali
    - Riferimenti
- Il meccanismo più sicuro è il primo: passare valori alle funzioni ed utilizzarne il valore di ritorno piuttosto che sfruttare effetti collaterali
  - Gli effetti collaterali vanno utilizzati solo quando costituiscono l'unica soluzione possibile, o quando l'altro meccanismo porterebbe ad un programma ancora più complesso

Direzione parametri

- Un altro aspetto importante nell'interazione tra parti di un programma è la direzione dei parametri delle funzioni
  - Si tratta di uno schema puramente concettuale, che poi si può realizzare, a seconda dei casi, con uno dei due tipi di passaggio (per valore o per riferimento) forniti dal linguaggio
  - Si considerano tipicamente tre possibilità:
    - parametri di ingresso
    - parametri di uscita
    - parametri di ingresso/uscita

Parametri di ingresso

- **Parametri di ingresso**: oggetti utilizzati solo per fornire i valori di ingresso su cui deve lavorare la funzione
  - Questo tipo di parametri si può implementare con il passaggio per valore o, come stiamo per vedere, con il passaggio per riferimento ad oggetto costante
- Prima di questa lezione abbiamo utilizzato solo parametri di ingresso

Parametri di uscita

- **Parametri di uscita**: oggetti che non saranno mai usati in lettura, ma solo per memorizzare dei valori di ritorno
  - Permettono di fatto di definire funzioni con un vettore di valori di ritorno
    - Un valore (magari quello più importante) viene ritornato mediante l'istruzione **return**
    - Gli altri si memorizzano nei parametri di uscita
- I parametri di uscita possono essere realizzati in modo naturale con il passaggio per riferimento

Esempio

```
/*
 * x : parametro di ingresso
 * y : parametro di uscita
 * La funzione ritorna true solo se il parametro di
 * uscita è da ritenersi valido
*/
bool fun(int x, int &y)
{
   if (x < 0)
       return false ;
   y = 3 ;
   return true;
}

main()
{
   int a, b ; cin>>b ;
   if (! fun(b, a))
       cout<<"Assegnamento di a non realizzato"<<endl ;
   else
       cout<<"a vale "<<a<<endl ;
}
```

Parametri di ingresso/uscita

- **Parametri di ingresso/uscita**: oggetti usati sia come parametri di ingresso che come parametri di uscita
- Realizzabili mediante passaggio per riferimento
- Tra i tre, sono probabilmente il tipo di parametro che porta alla peggiore leggibilità

Esercizio

- Definire una funzione che prenda in ingresso un oggetto di tipo carattere e, se si tratta di una lettera minuscola, lo trasformi nella corrispondente lettera maiuscola. La funzione deve ritornare vero se la conversione è avvenuta, falso altrimenti.
  - Tipo del parametro?
  - Tipo del valore di ritorno?
  - Direzione del parametro?
    - Forse ingresso/uscita?
- Per il calcolo del carattere in uscita, fate pure riferimento alla corrispondente funzione che abbiamo già realizzato come esercizio finale sui caratteri

Soluzione

- La direzione del parametro **c** è ingresso/uscita:

```
/*
 * Dato il carattere letto 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 ;
}
```

Realizzazione riferimenti

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

Costo passaggio per valore

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

Domanda

- Il costo del passaggio per riferimento aumenta all'aumentare delle dimensioni dell'oggetto passato?

Costo passaggio per riferimento

- 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 `const`

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

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

- Pertanto, mediante il passaggio per riferimento ad oggetto costante possiamo realizzare parametri di ingresso a **costo costante** (indipendente dalle dimensioni dell'oggetto passato)

Tipo di ritorno riferimento

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

Esempio

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

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

Cosa stampa?

Risposta

- Stampa
  11 11
- La funzione **`fun`** infatti ritorna un riferimento al proprio parametro formale
  - Ma il proprio parametro formale è a sua volta un riferimento al parametro attuale passato alla funzione
  - In conclusione la funzione ritorna un riferimento al parametro attuale che le si passa quando la si invoca
- Nel **`main`**, si passa come paramentro attuale la variabile **`c`**, pertanto il riferimento **`b`** viene inizializzato con l'indirizzo della variabile **`c`**

Esempio commentato

```
int & fun(int &a)
{
      return a ; // ritorna un riferimento al parametro
                 // formale a, ma il parametro formale a
                 // è a sua volta un riferimento;
                 // siccome tale riferimento sarà
                 // inizializzato con l'indirizzo del
                 // parametro attuale all'atto della
                 // chiamata, la funzione ritorna in
                 // definitiva un riferimento al parametro
                 // attuale stesso
}

main()
{
      int c = 10 ;
      int &b = fun(c) ; // b è inizializzato all'indirizzo di c
      b++ ;          // incrementando b, si incrementa di fatto c
      cout<<b<<" "<<c<<endl ;
}
```

Esempio

```
int a = 10 ;

int & fun()
{
     return a ;
}

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

Cosa stampa?

Risposta

- Stampa
  11 11
- La funzione **`fun`** infatti ritorna un riferimento alla variabile globale **`a`**
- Utilizzando il riferimento ritornato dalla funzione, nel **`main`** il riferimento **`b`** viene inizializzato con l'indirizzo della variabile globale **`a`**

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**?

Errore

- Ritornare un riferimento ad una variabile locale è un **errore logico**
  - Non ha senso ritornare un riferimento ad un oggetto che non esiste più
- E' anche un **errore di gestione della memoria**
  - Ci si riferisce ad una zona di memoria non più correttamente allocata per contenere un oggetto del programma
  - Il compilatore può non segnalare l'errore
  - Se poi si effettuano assegnamenti tramite tale riferimento si può corrompere la memoria del programma

Passaggio per riferimento in C

- Come già detto, il passaggio per riferimento è disponibile in molti linguaggi di alto livello
  - C++, Pascal, Java, ...
- Però NON è disponibile in C, e quindi in C deve essere emulato tramite l'uso di **puntatori**
  - In questa lezione non vedremo ulteriori dettagli in merito