# Protezione della Ereditarietà

# Protezione di ereditarietà

- L'eredità in C++ può essere di tre tipi:
  - pubblica (**`public`**)
  - privata (**`private`**)
  - protetta (**`protected`**)

- Si distingue in base alle restrizioni di accesso che si impongono su i dati membro e sulle funzioni membro ereditate

# Significato intuitivo

- Ereditare in modo pubblico significa che ciò che prima era visibile all'esterno rimane visibile all'esterno anche dopo l'eredità
- Ereditare in modo protetto o privato significa che ciò che prima era visibile all'esterno rimane visibile solo all'interno della classe che eredita

# Eredità pubblica:

- Ciò che è pubblico nella classe base *è* accessibile alla classe derivata ed *è* accessibile all'esterno
- Ciò che è protetto nella classe base *è* accessibile alla classe derivata ma *non è* accessibile all'esterno
- Ciò che è privato nella classe base *non è* accessibile alla classe derivata e *non è* accessibile all'esterno

# Eredità pubblica:

```
Class Base{
public:
   Base(int usr_a=0, int usr_b=0, int usr_c=0)
   {a_pub=usr_a;b_pro=usr_b;c_pri=usr_c;}
   int a_pub;
protected:
   int b_pro;
private:
   int c_pri;
};
```

# Eredità pubblica:

```
class Derivata: public Base{
public:
   Derivata(int usr_a=0, int usr_b=0, int usr_c=0, int
   usr_a_d=0, int usr_c_d=0)
   :Base(usr_a,usr_b,usr_c){a_d=usr_a_d;c_d=usr_c_d;}
   int get_a(){return a_pub;}
   int get_b(){return b_pro;}
   int get_c(){return c_pri;}
   int a_d_pub;
private:
   int c_d_pri;
};
```

# Eredità pubblica:

```
Void main(){
   Derivata obj(1,2,3,4,5);
   //visibilità dal main
   obj.a_pub=10;//OK
   obj.b_pro=10;//NO
   obj.c_pri=10;//NO
   obj.a_d_pub=10;//OK
   obj.c_d_pri=10;//NO

   //visibilità nella classe derivata
   obj.get_a();//OK
   obj.get_b();//OK
   obj.get_c();//NO la classe derivata non ha accesso al dato
   privato della classe base
}
```

# Eredità protetta:

- Ciò che è pubblico nella classe base *è* accessibile alla classe derivata ma *non è* accessibile all'esterno

  (ma può essere tramandato ai discendenti in modo che questi possano accedervi)
- Ciò che è protetto nella classe base *è* accessibile alla classe derivata ma *non è* accessibile all'esterno

- Ciò che è privato nella classe base *non è* accessibile alla classe derivata e *non è* accessibile all'esterno

# Eredità protetta:

```
Void main(){
   Derivata obj(1,2,3,4,5);

   obj.a=10;//NO
   obj.b=10;//NO
   obj.c=10;//NO
   obj.a_d=10;//OK
   obj.c_d=10;//NO

   obj.get_a();//OK
   obj.get_b();//OK
   obj.get_c();//NO
}
```

# Eredità privata:

- Ciò che è pubblico nella classe base *è* accessibile alla classe derivata e *non è* accessibile all'esterno
  (né è accessibile ai discendenti)
- Ciò che è protetto nella classe base *è* accessibile alla classe derivata ma *non è* accessibile all'esterno

- Ciò che è privato nella classe base *non è* accessibile alla classe derivata e *non è* accessibile all'esterno

# Eredità privata:

```
Void main(){
   Derivata obj(1,2,3,4,5);

   obj.a=10;//NO
   obj.b=10;//NO
   obj.c=10;//NO
   obj.a_d=10;//OK
   obj.c_d=10;//NO

   obj.get_a();//OK
   obj.get_b();//OK
   obj.get_c();//NO
}
```

# Eredità protetta e privata:

- I membri pubblici e protetti della classe base se ereditati in modo protetto diventano protetti nella classe derivata

- I membri pubblici e protetti della classe base se ereditati in modo privato diventano privati nella classe derivata

# Eredità protetta e privata:

- La differenza fra l'ereditarietà protetta o privata è rilevabile solo se la classe derivata è a sua volta ereditata da una o più classi gerarchicamente
- Nel caso di eredità protetta i membri pubblici e protetti rimangono accessibili per tutta la gerarchia (ma non all'esterno)
- Mentre nel caso di eredità privata i membri di qualsiasi tipo cessano di essere trasmessi in modo accessibile ai discendenti

# Utilita' eredità privata

- Data una classe ListClass che manipola le liste si può derivare una classe QueueClass che la eredita in modalità privata
- Tutti i metodi di ListClass diventano privati di QueueClass e non accessibili all'esterno
- Si definiscono i metodi pubblici di QueueClass che non fanno altro che richiamare i metodi di ListClass di interesse
  - ex: enqueue chiama insertAtBack e dequeue chiama removeFromFront
- ..gli altri metodi rimangono inaccessibili

# Costruttori in classi derivate

- Dato che una classe derivata contiene i membri della classe base quando viene istanziata deve poter accedere al costruttore della classe base

- Sintassi:

```
ClasseDerivata(arg_base, arg_deriv)
:ClasseBase(arg_base){...}
```

- Se non viene fatto in modo esplicito allora la classe derivata chiama in modo implicito il costruttore di default della classe base

# Distruttori in classi derivate

- Dato che il distruttore viene invocato automaticamente e non prende parametri, la classe derivata non ha un modo esplicito per invocare il distruttore della classe base
- Non si deve fare niente per i distruttori, vengono chiamati automaticamente nell'ordine giusto

# Ordine di chiamata dei Costruttori/Distrutt

```
#include<iostream>
class Base{
public:
   Base(){cout<<"Costruzione Base\n";}
   ~Base(){cout<<"Distruzione Base\n";}
};
class Deriv1:public Base{
public:
   Deriv1(){cout<<"Costruzione Deriv1\n";}
   ~Deriv1(){cout<<"Distruzione Deriv1\n";}
};
class Deriv2:public Deriv1{
public:
   Deriv2(){cout<<"Costruzione Deriv2\n";}
   ~Deriv2(){cout<<"Distruzione Deriv2\n";}
};
```

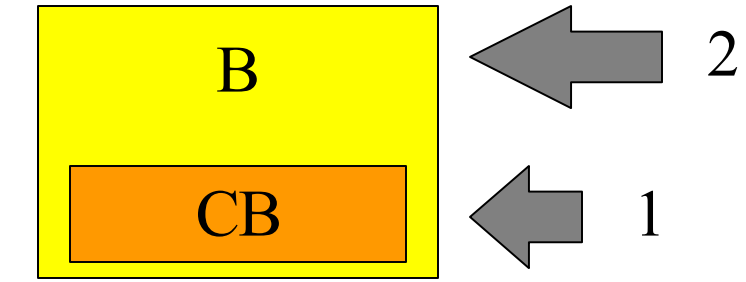
```
void main(){
   Deriv2 ob;
}
```

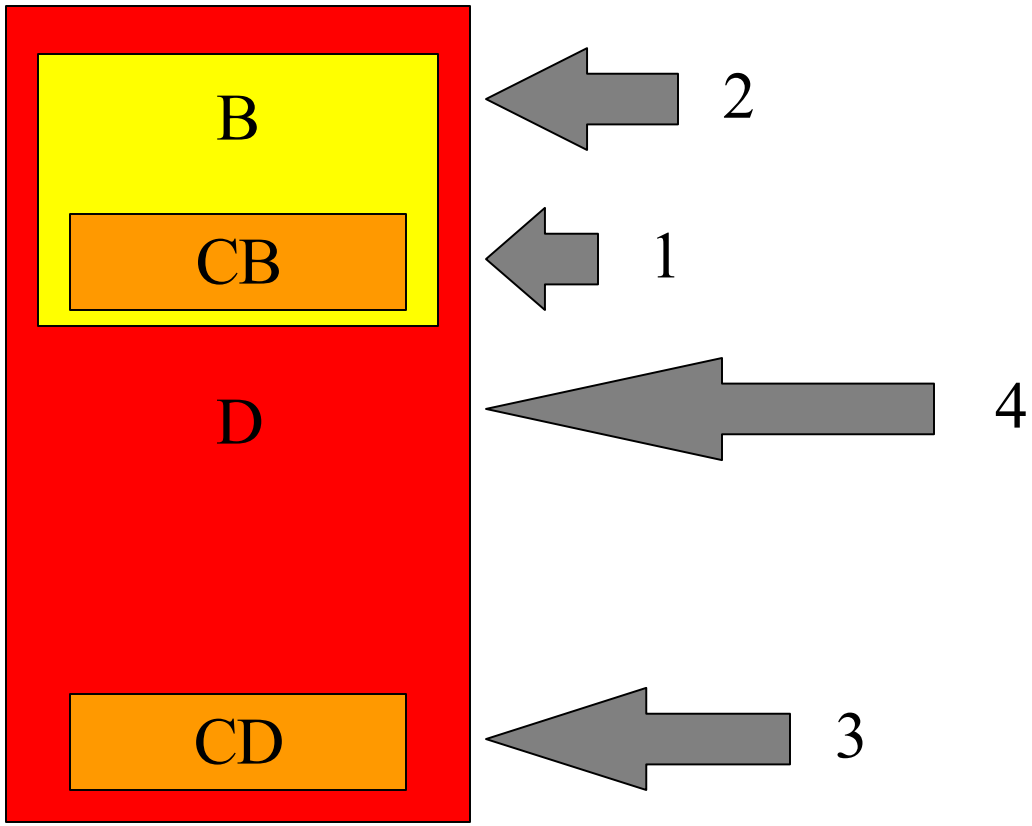
```
Output:
Costruzione Base
Costruzione Deriv1
Costruzione Deriv2
Distruzione Deriv2
Distruzione Deriv1
Distruzione Base
```

# Nota sui costruttori

- Si ha un oggetto di una classe derivata D da una classe base B. Sia B che D contengono oggetti di altre classi CB e CD rispettivamente
- Quando si crea un oggetto di tipo D sono eseguiti prima i costruttori degli oggetti CB poi il costruttore di B, poi i costruttori degli oggetti CD e infine il costruttore di D
- I distruttori sono chiamati in ordine inverso rispetto ai corrispondenti costruttori

# Ordine di invocazione dei costruttori



# Eredità singola o multipla

- Ereditarietà singola: la classe derivata eredita solo da una classe base
- Ereditarietà multipla: la classe derivata eredita da più classi base (anche fra di loro eterogenee)

# Eredità multipla

```
class Base1{
public:
   Base1(int i=0){b1=i;}
private:
   int b1;
};

class Base2{
public:
   Base2(int i=0){b2=i;}
private:
   int b2;
};

class Deriv:public Base1, public Base2{
public:
   Deriv(int i=0, int j=0, int z=0):Base1(i),
   Base2(j){d=z;}
private:
   int d;
};
```

# Relazioni fra classi

- Due classi possono essere in relazione l'una con l'altra nei modi:
  - **è un**
  - **ha un**

# Relazione: è un

- Definizione: una classe è una specializzazione di una seconda classe
- Implementazione: tramite il meccanismo di ereditarietà
- Es: una classe cerchio è una classe punto (a cui si è aggiunto dati e membri)

# Relazione: ha un

- Definizione: una classe ha nella sua composizione altre classi
- Implementazione: una classe ha fra i suoi dati membro altre classi
- Es: una classe Impiegato ha fra i propri attributi la classe Anagrafica e la classe Azienda