```cpp
/*

Esercizio 1

*/

/*

Chiamata A: (4) - (4) - (4)
Chiamata B: (4,7) - (4,7) - Chiamata Ambigua
Chiamata C: (1,2) - (1,2) - (1)
Chiamata D: (1,2,5) - (1,2,5) - (5)
Chiamata E: (1,2) - (1,2) - Chiamata Ambigua
Chiamata F: (1,2,5) - (1,2,5) - (2)
Chiamata G: (3) - (3) - (3)
Chiamata H: (3,6) - (3,6) - Chiamata Ambigua

*/



/*

Esercizio 2

*/

#include <iostream>
#include <algorithm>
#include <vector>
#include <iterator>

template <typename Iter, typename Pred>
unsigned count_if(Iter i1, Iter i2, Pred p) {

  // Uso unsigned per contenere solo valori positivi e lo zero.
  unsigned res=0;

  while(i1 != i2) {
    if(p(*i1))
      res++;

    ++i1;
  }

  return res;
}

int main() {

  std::vector<std::string> vs;
  // Uso un iteratore di input per leggere gli elementi successivi del contenitore dallo standard di input.
  std::istream_iterator<std::string> in(std::cin);
  // Iteratore per il controllo della fine della sequenza.
  std::istream_iterator<std::string> i_end;
  // Copio gli elementi letti nel range che parte da 'in' e arriva a 'i_end' dentro al range di 'back_inserter'.
  // 'back_inserter' inserisce i nuovi elementi in coda al vettore 'vi'.
  std::copy(in, i_end, std::back_inserter(vi));

  unsigned n;

  n = count_if(vs.begin(), vs.end(),
                                    // Inizio lambda expression.
                                    [](std::string s) {
                                      if (s.size() > 10) return true;
                                      else return false;
                                    }
                                    // Fine lambda expression.
                                  )

  std::cout << n << std::endl;
```

```
}


/*

Esercizio 3

*/

=== 1 ===
B::f(int)
B::g(int)
B::f(int)
C::g(int)
=== 2 ===
A::f(double)
B::f(int)
C::g(int)
=== 3 ===
B::f(int)
C::g(int)
=== 4 ===
Destructor C::~C()
Destructor B::~B()
Destructor A::~A()
Destructor B::~B()
Destructor A::~A()


/*

Esercizio 4

*/

/*

Il progetto in sè può andare bene.
Infatti notiamo come le classi abbiano quasi tutti metodi ben distinti che possono essere usati in modo specifico.
L'unica eccezione riguarda i tre metodi 'DoMsg1', 'DoMsg2' e 'DoMsg3' nelle due classi derivate.
Per questo è opportuno inserire all'interno della classe base tre metodi virtuali puri omonimi.
Questo serve per eseguire la risoluzione dell'overriding se e quando è necessario.

*/

class BasicProtocol {
private:
  /* ... */
public:
  BasicProtocol();
  virtual ~BasicProtocol();

  bool BasicMsgA( /* ... */ );
  bool BasicMsgB( /* ... */ );
  bool BasicMsgC( /* ... */ );

  virtual bool DoMsg1( /* ... */ ) = 0;
  virtual bool DoMsg2( /* ... */ ) = 0;
  virtual bool DoMsg2( /* ... */ ) = 0;
};

class Protocol1 : public BasicProtocol {
public:
  Protocol1();
  ~Protocol1();

  bool DoMsg1( /* ... */ );
  bool DoMsg2( /* ... */ );
  bool DoMsg3( /* ... */ );
```

```
};

class Protocol2 : public BasicProtocol {
public:
  Protocol2();
  ~Protocol2();

  bool DoMsg1( /* ... */ );
  bool DoMsg2( /* ... */ );
  bool DoMsg3( /* ... */ );
  bool DoMsg4( /* ... */ );
  bool DoMsg5( /* ... */ );
};


/*

Esercizio 5

*/

/*

Il primo problema, che salta subito all'occhio, è che i dati (risorse) non sono protetti.
Bisogna infatti renderli privati utilizzando 'private'.
Il secondo problema è sul costruttore di A.
Infatti bisogna capire cosa succede se le due 'new' falliscono.
Se fallisce la prima non è un problema dato che non è stata acquisita ancora nessuna risora.
Ma se la prima ha successo e la seconda lancia un'eccezione abbiamo un problema.
Infatti noi usciremmo da A in modo eccezionale senza averlo costruito ma acendo acquisito 'pi'.
Serve quindi usare un try-catch dicendo che inizialmente il secondo puntatore non contiene nessuna risorsa.

*/


#include <string>

class A {

private:
 int* pi;
 std::string str;
 double* pd;

 // Dichiaro le seguenti funzioni per evitare i costruttori di copia e spostamento.
 A(const A&); // Privato non implementato.
 operator=(const A&); // Privato non implementato.

public:
  A(const std::string& s) : pi(new int), str(s), pd(new double) {
    try { // Proteggo la risorsa *pi già acquisita.
      pd = new double;
    }
    catch(...) {
      delete pi;
      throw;
    }
  }
  ~A() { delete pi; delete pd; }

};
```