Esercizio Funzione Definire un template di funzione `template<class T> list<const iostream*> fun(vector<ostream*>&)` con il seguente comportamento: in ogni invocazione `fun(v)`, per ogni puntatore `p` elemento (di tipo `ostream*`) del vector `v`:

1. se `p` non è nullo e `*p` è un `fstream` che non è nello stato good (ovvero stato 0, con tutti i bit di errore spenti), allora `p` diventa nullo;
2. se `p` non è nullo e `*p` è uno `stringstream` nello stato good, allora `p` viene inserito nella lista che la funzione deve ritornare;
3. se la lista che la funzione deve ritornare è vuota allora la funzione solleva una eccezione di tipo `T`, altrimenti ritorna la lista.

```
template<class T> list<const iostream*> fun(vector<ostream*>& v){
        list<const iostream*> ret;

        for(auto p = v.begin(); p != v.end(); ++p){
                if(p != nullptr && !(dynamic_cast<fstream*>(*p))->good()){
                        p = nullptr;
                }
                if(p != nullptr && (dynamic_cast<stringstream*>(*p))->good()){
                        ret.push_back(p);
                }
        }
        if(ret.isEmpty()) throw T();
        return ret;
}
```

Esercizio Funzione.

Definire un template di funzione

`template <class T> list<const iostream*> compare(vector<ostream*>&, vector<const T*>&)`

con il seguente comportamento: in ogni invocazione `compare(v,w)`,

1. se `v` e `w` non contengono lo stesso numero di elementi allora viene sollevata una eccezione di tipo `string` che rappresenta la stringa vuota;
2. se `v` e `w` contengono lo stesso numero di elementi allora per ogni posizione `i` dentro i bounds dei due vettori `v` e `w`:
   (a) se `*v[i]` è un `fstream` ed è dello stesso tipo di `*w[i]` allora: (i) il puntatore `v[i]` viene inserito nella lista che la funzione deve ritornare; (ii) i puntatori `v[i]` e `w[i]` vengono rimossi dai vettori che li contengono;
   (b) se `*w[i]` è uno `stringstream` in stato good e `*v[i]` e `*w[i]` sono di tipo diverso allora il puntatore `w[i]` viene inserito nella lista che la funzione deve ritornare.

```
template <class T> list<const iostream*> compare(vector<ostream*>& v, vector<const T*>& w){
        list<const iostream*> ret;

        // SE non usi "auto" allora....
        // for(vector<ostream*>::iterator it = ....
        // for(vector<const T*>::const_iterator cit = ....

        for(int i = 0, j = 0; i < v.size() && j < w.size; ++i, j++){
                if(v.size() != w.size()) throw std::string("");

                fstream* f = dynamic_cast<fstream*>(*v[i]);
                if(f && typeid(f) == typeid(*w[i])){
                        ret.push_back(f);

                        // Per ostream, no const -> puoi fare direttamente erase
                        v.erase(v[i]);

                        // Per "w"; const e quindi copia per togliere il const ed erase
                        T* t = w[i];
                        v.erase(t);
                        delete t;
                }
                stringstream* s = dynamic_cast<stringstream*>(*w[i]);
                if(s && s->good() && typeid(s) != typeid(f)){
                        ret.push_back(s);
                }
        }

        return ret;
}
```