```cpp
template <class T>
vector<const T*> filter(const list<ios*>& streams, const function<bool(const T*)>& predicate) {
    // Verifica che T sia un sottotipo di ios
    if (!is_base_of<ios, T>::value) {
        throw std::logic_error("Tipo non compatibile");
    }

    vector<const T*> result;

    // Itera sulla lista di streams
    for (auto it = streams.begin(); it != streams.end(); ++it) {
        // Prova a fare il cast a T*
        if (const T* derived = dynamic_cast<const T*>(*it)) {
            // Verifica se l'elemento soddisfa il predicato
            if (predicate(derived)) {
                result.push_back(derived);
            }
        }
    }

    // Se non sono stati trovati elementi che soddisfano il predicato, lancia un'eccezione
    if (result.empty()) {
        throw ios_base::failure("Nessun elemento trovato");
    }

    return result;
}
```

Spiegazione

La soluzione implementa il template di funzione `filter` seguendo i requisiti specificati:

1. Prima di tutto, verifica che `T` sia effettivamente un sottotipo di `ios` usando `std::is_base_of`. Se non lo è, lancia un'eccezione `std::logic_error`.
2. Crea un vector vuoto `result` che conterrà i puntatori agli elementi filtrati.
3. Itera sulla lista di streams:
   - Per ogni elemento, prova a convertirlo al tipo `const T*` usando `dynamic_cast`
   - Se la conversione ha successo, verifica se l'elemento soddisfa il predicato fornito
   - Se entrambe le condizioni sono soddisfatte, aggiunge il puntatore al vector risultato
4. Alla fine, se non sono stati trovati elementi che soddisfano il predicato (il vector risultato è vuoto), lancia un'eccezione `ios_base::failure`.
5. Altrimenti, restituisce il vector con i puntatori agli elementi filtrati.

Questa implementazione gestisce correttamente tutti i requisiti dell'esercizio ed è sicura per l'uso con qualsiasi tipo di stream derivato da `ios`.