2022-07-05

Definire la funzione generica all_of che prende in input una sequenza ed un predicato unario e restituisce in output un booleano: la funzione restituisce il valore true se e solo se tutti gli elementi della sequenza soddisfano il predicato. Scrivere inoltre una funzione che prende in input un vettore di numeri interi e, usando la funzione all_of appena definita, controlla se tutti gli interi contenuti nel vettore sono negativi.

```
#include <vector>
#include <iostream>
template <typename Type, typename PredUn>

bool all_of(Type first, Type last, PredUn pred) {
  for (it = first; it != last; ++it) {
    if (!pred(*it)) {
      return false;
    }
  }
  return true;
}

bool tuttiNegativi(vector<int>& vec) {
  return all_of(vec.begin(), vec.end(), [](int x) { return x < 0; });
}
```

2012-02-02

2. Si forniscano il prototipo e l'implementazione della funzione generica `count_if`, che dati in ingresso una sequenza ed un predicato unario, restituisce il numero di elementi di quella sequenza per i quali quel predicato è soddisfatto.

   Utilizzando la funzione suddetta, scrivere un programma che legge dallo standard input una sequenza di `std::string` e scrive sullo standard output il numero di stringhe lette aventi una lunghezza maggiore di 10.

```
#include <iostream>
#include <vector>

template <typename Type, typename PredUn>

int count_if(Type first, Type last, PredUn pred) {
	int c = 0;
	for (auto it = first; first != last; it++) {
		if (pred(*it))
			c++;
	}
	return c;
}

int stringhe(vector<string>& vec) {
	return count_if(vec.begin, vec.end, [](string& x){ x.length > 10 })
}
```

2010-06-14 da fare

5. Si forniscano il prototipo e l'implementazione della funzione generica `transform` i cui parametri specificano: (a) due sequenze di ingresso (non necessariamente dello stesso tipo), di cui si assume che la seconda sia lunga almeno quanto la prima; (b) una sequenza di uscita (potenzialmente di un tipo ancora diverso); (c) una funzione binaria applicabile a coppie di elementi in ingresso (il primo preso dalla prima sequenza, il secondo dalla seconda) e che restituisce un elemento assegnabile alla sequenza di uscita.

   La funzione `transform` applica la funzione binaria agli elementi in posizione corrispondente nelle due sequenze di ingresso, assegnando il risultato ad elementi consecutivi della sequenza di uscita.

   Utilizzando la funzione suddetta, scrivere una funzione che, date due liste di interi $[i_1, \ldots, i_n]$ e $[j_1, \ldots, j_n]$ scrive sullo standard output la sequenza delle medie aritmetiche (di tipo `double`) $[(i_1 + j_1)/2, \ldots, (i_n + j_n)/2]$.

```
#include <vector>
using namespace std
template<typename In1, typename In2, typename Out, typename PredBin>

auto transform (In1 in1_i, In1 in1_f, In2 in2, Out out, PredBin predbin) {
        //predicato di output == controllo predicato di input1 e input2
}
```

2008-06-16

4. Scrivere l'implementazione dell'algoritmo generico della STL `max_element`, che prende come input una sequenza ed un criterio di ordinamento (un predicato binario) e restituisce l'iteratore all'elemento massimo contenuto nella sequenza (comportandosi in modo standard nel caso di sequenza vuota). Elencare in modo esaustivo i requisiti imposti dall'implementazione sui parametri di tipo e sugli argomenti della funzione. In particolare, individuare le categorie di iteratori che *non* possono essere utilizzate per istanziare il template, motivando la risposta.

```
#include <iostream>
#include <vector>

template<typename Type, typename PredBin>

Type max_element (Type First, Type Last, PredBin Bin) {
	if (first == Last)
		return Last;

	auto max = First;

	for (auto it = ++First; it != Last; it++) {
		if (Bin(*max, *it))
			max = it;
	}
	return max
}
```

2006-02-27

5. Scrivere l'implementazione dell'algoritmo della STL `remove_copy_if`, che ha come parametri una sequenza di input ed una sequenza di output, specificate mediante iteratori, ed un predicato unario. L'algoritmo copia nella sequenza di output, mantenendone l'ordine, gli elementi della sequenza di input che *non* soddisfano il predicato; l'algoritmo restituisce l'iteratore corrispondente alla fine della sequenza di output.

   Utilizzando l'algoritmo precedente, implementare l'analogo algoritmo `remove_if`, che invece di scrivere su di una sequenza di output modifica direttamente la sequenza di input. Motivare brevemente quali siano le categorie di iteratori utilizzabili in ognuno dei due algoritmi.

```
#include <vector>
using namespace std
template<typename Type1, typename Type2, typename PredUn>

Type2 remove_copy_if(const Type1& seq1, Type2& seq2, PredUn pred) {
	for(auto it1 = seq1; it1 != seq1.end(); it++) {
		if(!pred(*it1)) {
			seq2.push_back(*it1);
			//per aggiungere in una posizione qualsiasi si usa:
			// seq2.insert(seq2.end(), *it1);}
	}
	return seq2.end();
}

void remove_if(Type1& seq1, Type2& seq1, PredUn pred) {
	remove_copy_if(seq1, seq1, pred);
}
```

2006-02-06

5. Scrivere l'implementazione dell'algoritmo della STL `find_first_of`. L'algoritmo prende in input due sequenze `[first1, last1)` e `[first2, last2)`, specificate mediante iteratori, ed un predicato binario `comp`. Utilizzando il predicato come operatore di confronto, l'algoritmo cerca nella prima sequenza il primo elemento "uguale" ad un elemento qualunque della seconda sequenza. Se lo trova, restituisce l'iteratore corrispondente; altrimenti, restituisce `last1`. Si noti che le due sequenze possono avere tipi diversi. Indicare, motivando la risposta, la più ampia categoria di iteratori utilizzabile per ognuna delle due sequenze di input.

```
#include <vector>
using namespace std
template<typename Type1, typename Type2 typename PredBin>

Type1 find_first_of (Type1 first1, Type1 last1, Type2 first2, Type2 last2, PredBin comp) {
        for(auto it1 = first1; it != last1; it1++) {
                for(auto  it2 = first2; it2 != last2; it2++)
                        if(comp(*it1, *it2)) {
                                return it1;
                        }
        }
        return last1;
}
```