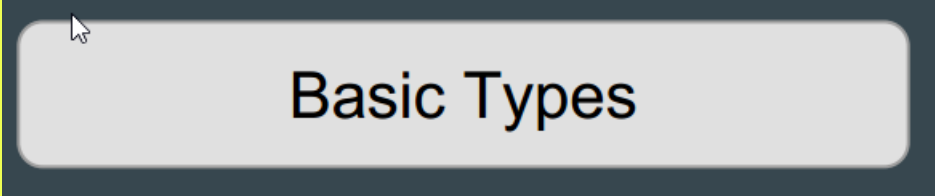


C++: STL: ALGORITHM

STL: ALGORITHM

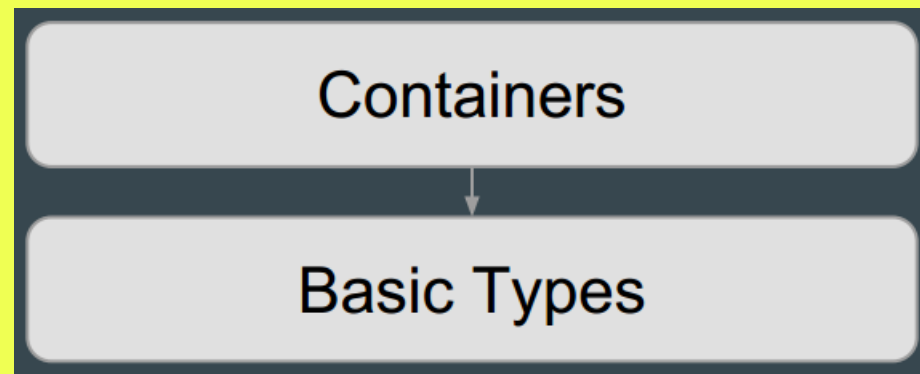
- ▶ Punto della situazione – in maniera astratta
- ▶ Base di C e C++ di sono i basic types: char, int, float, double, ecc...
Ogni elemento di questi tipi, concettualmente contiene un valore
- ▶ C e C++ permette la creazione di tipi complessi attraverso struct e oggetti



Basic Types

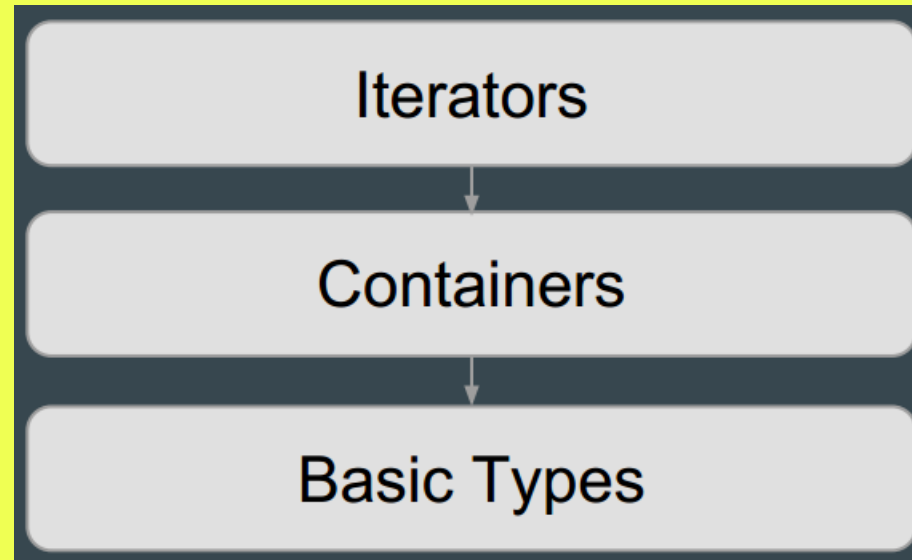
STL: ALGORITHM

- ▶ Molti programmi richiedono un numero di variabili dello stesso tipo
 - es.: un vettore di interi che rappresenta l'età degli studenti
- ▶ I container permettono di usare lo stesso tipo di raggruppamento a prescindere dal basic type utilizzato
 - la stessa implementazione di `<vector>` può essere usata nello stesso modo per `int` o `string`



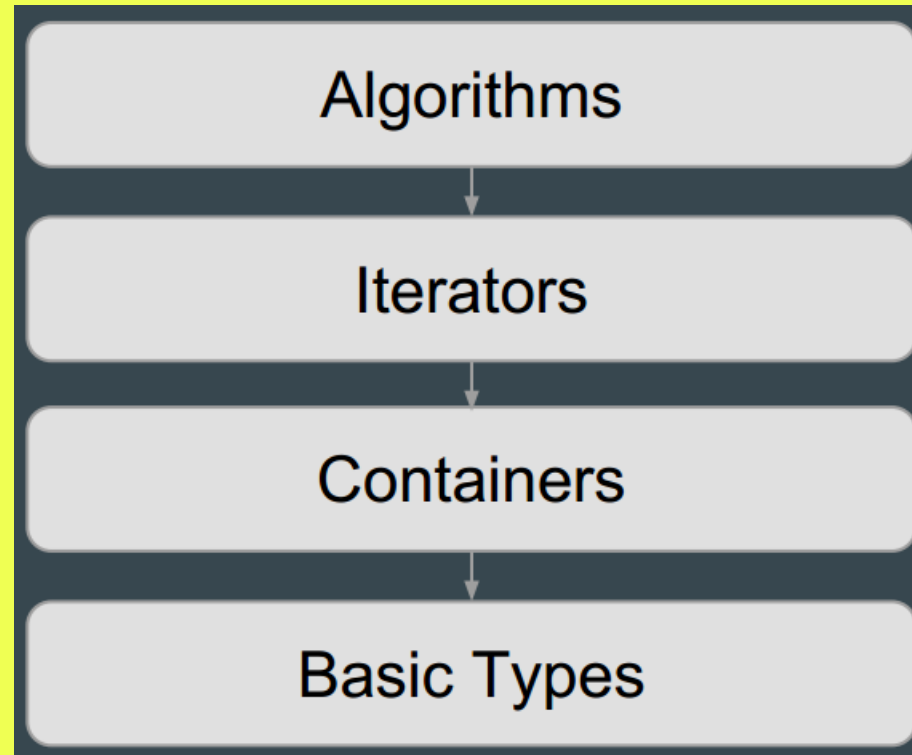
STL: ALGORITHM

- ▶ Questo ci permette di usare i container per eseguire svariate operazioni su un basic type, a prescindere da quale esso sia.
- ▶ Saliamo di un gradino: è possibile eseguire svariate operazioni sui container a prescindere da quali essi siano?
- ▶ Iterator permettono di astrarre il tipo di container utilizzato



STL: ALGORITHM

- ▶ Saliamo ancora. È possibile astrarre svariate operazioni sugli iterator (e quindi sui container), a prescindere da quali essi siano? Operazioni come ordinamento, ricerca, ecc, possono essere scritte ad-hoc per ogni tipo di container
- ▶ C++ mette a disposizione <algorithm>



STL: ALGORITHM

- ▶ Insieme di funzioni specificatamente progettate per operare su un **range di elementi**
- ▶ Range di elementi è una sequenza di oggetti a cui si possa accedere tramite iteratori o puntatori, per esempio un array o qualsiasi container dalla STL.
- ▶ Le prime librerie includevano queste funzioni all'interno dei container stessi. STL separa gli algoritmi dai container per il concetto di astrazione che abbiamo appena illustrato.
- ▶ Quanti sono e cosa fanno?
- ▶ <http://www.cplusplus.com/reference/algorithm/>

STL: ALGORITHM

- ▶ Sintassi comune (quasi sempre) per molte delle funzioni
- ▶ Accettano un range sotto forma di intervallo tra puntatori su cui operare
- ▶ Es:
- ▶

```
template <class InputIterator, class T>
    typename iterator_traits<InputIterator>::difference_type
    count (InputIterator first, InputIterator last, const T& val);
```
- ▶

```
std::vector<int> v1 = {3,6,0,3,5,0,1,4};
int c1 = std::count(v1.begin(), v1.end(), 0);
```

STL: ALGORITHM

- ▶ Sintassi comune (quasi sempre) per molte delle funzioni
- ▶ Accettano un range sotto forma di intervallo tra puntatori su cui operare
- ▶ Es:
 - ▶

```
template <class InputIterator, class T>
    typename iterator_traits<InputIterator>::difference_type
    count (InputIterator first, InputIterator last, const T& val);
```
- ▶

```
std::vector<int> v1 = {3,6,0,3,5,0,1,4};
int c1 = std::count(v1.begin(), v1.end(), 0);
```
- ▶ Ma anche...
- ▶

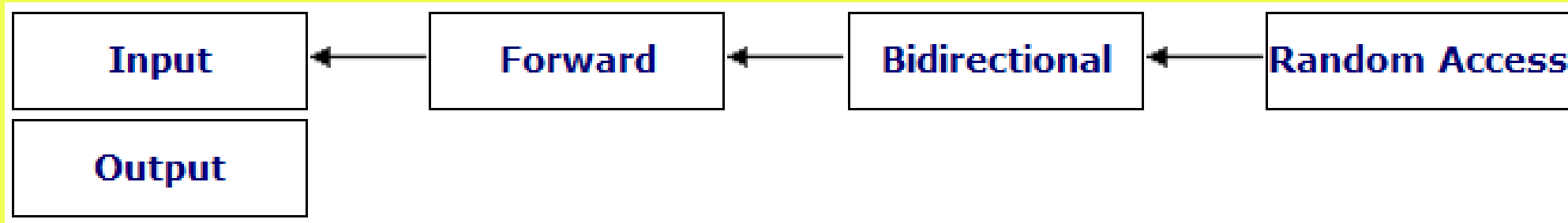
```
int * v2 = new int[8] {3,6,0,3,5,0,1,4};
int c2 = std::count(v2, v2 + 8, 0);
```


STL: ALGORITHM

- ▶ Storicamente: in C abbiamo i puntatori e questo concetto viene adattato in C++ dalla sua nascita.
- ▶ Fine anni '90: idea chiamata «iterator concept» introdotta in C++. Iterator concept legato a STL e generic programming. Puntatore in C rappresenta, per es., una locazione in un array => iterator rappresenta una locazione in un container.
- ▶ Idea semplificata: se un tipo di dato supporta una lista di operazioni e comportamenti tali per cui questo rappresenta una locazione in un container, e prevede l'accesso all'elemento del container in quella locazione, questo tipo di dato può venir chiamato iterator.
- ▶ I puntatori in C aderiscono a questa definizione tramite gli operatori [], & e *, quindi i puntatori sono a tutti gli effetti una classe di iteratori.

STL: ALGORITHM

- ▶ Tipi di iteratori: anche in questo caso, previste diverse astrazioni che ne modellizzano il comportamento



- ▶ <http://www.cplusplus.com/reference/iterator/>
- ▶ C pointer a quale categoria appartengono?
- ▶ Quando creiamo un iteratore a partire da un container, di quale tipo sarà?
- ▶ <http://www.cplusplus.com/reference/vector/vector/>
http://www.cplusplus.com/reference/forward_list/forward_list/

STL: ALGORITHM

- ▶ Insieme di funzioni specificatamente progettate per operare su un range di elementi
- ▶ Range di elementi è una sequenza di oggetti a cui si possa accedere tramite iteratori o puntatori, per esempio un array o qualsiasi container dalla STL.
- ▶ Le prime librerie includevano queste funzioni all'interno dei container stessi. STL separa gli algoritmi dai container per il concetto di astrazione che abbiamo appena illustrato.
- ▶ Questo complicato modello di astrazione serve a:
 - non scrivere codice duplicato
 - scrivere codice più corretto e più efficiente
 - scrivere codice più conciso, più leggibile e più mantenibile

STL: ALGORITHM

- ▶ Esempio: algoFun.cpp