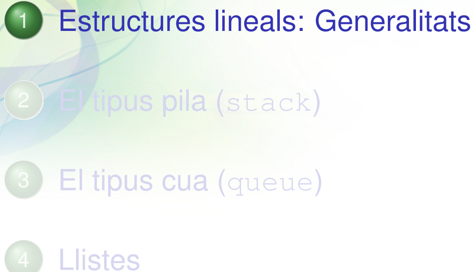
Estructures lineals: Piles, Cues i Llistes
Programació 2
Facultat d'Informàtica d'Informàtica, UPC

Professorat de PRO2

Tardor 2022

- Col·laboracions (en ordre alfabètic): Juan Luis Esteban, Ricard Gavaldà, Conrado Martínez, Fernando Orejas
- Aquestes transparències **no** substitueixen els apunts de l'assignatura, els complementen

Part I



1. Estructures lineals: Generalitats
2. El tipus pila (`stack`)
3. El tipus cua (`queue`)
4. Llistes

Estructures lineals: Generalitats

Una estructura lineal C és un conjunt d'elements d'un cert tipus `T`

$$C = [a_1, a_2, \ldots, a_n]$$

en el que es defineix una relació de successió

- Per tot i, $1 \leq i < n$, a_{i+1} és el successor de a_i. L'últim element a_n no té successor.
- Per tot i, $1 < i \leq n$, a_{i-1} és el predecessor de a_i. El primer element a_1 no té predecessor.

Si $n = 0$ la estructura està buida

Estructures lineals: Generalitats

- Piles (*stack*)
- Cues (*queue*)
- Llistes (*list*)

Part I

1 Estructures lineals: Generalitats

2 El tipus pila (`stack`)

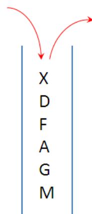
3 El tipus cua (`queue`)

4 Llistes

La classe `stack`

Ofereix tres operacions bàsiques:

- Afegir un nou element al final (*empilar*)
- Treure l'últim element (*desempilar*)
- Examinar l'últim element (*cim*)



LIFO - *Last In, First Out*: el darrer que ha entrat serà el primer en sortir, i és l'únic accessible

Part I

1 Estructures lineals: Generalitats

2 El tipus pila (`stack`)

3 El tipus cua (`queue`)

4 Llistes

La classe `queue`

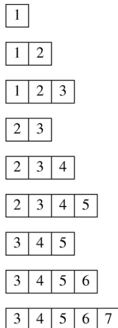
Ofereix tres operacions bàsiques:

- Afegir un nou element al final (*encuar*)
- Treure el primer element (*desencuar*)
- Examinar el primer element (*front*)

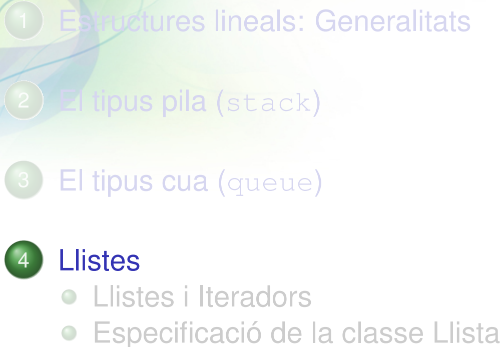
FIFO - *First In, First Out*: el primer que ha entrat serà el primer en sortir i és l'únic accessible

Exemple d'evolució d'una cua

```
queue<int> c;
c.push(1); c.push(2); c.push(3);
c.pop();
c.push(4); c.push(5);
c.pop();
c.push(6); c.push(7);
```



Part I



1. Estructures lineals: Generalitats
2. El tipus pila (`stack`)
3. El tipus cua (`queue`)
4. Llistes
   - Llistes i Iteradors
   - Especificació de la classe Llista

- Exemples d'operacions amb llistes
- Splice
- Fusió ordenada
- Creació de llistes

Part I

1. Estructures lineals: Generalitats
2. El tipus pila (`stack`)
3. El tipus cua (`queue`)
4. Llistes
   - Llistes i Iteradors
   - Especificació de la classe Llista
   - Exemples d'operacions amb llistes
   - Splice
   - Fusió ordenada
   - Creació de llistes

Llistes

Les llistes ens ofereixen operacions per a fer:

- Recorreguts seqüencials de tots els elements
- Inserció d'un element nou a qualsevol punt de la seqüència
- Eliminació d'un element qualsevol
- Concatenació

Iteradors

- El mecanisme que permet fer això amb les `list` de la STL són els iteradors
- Un *iterador* és un objecte que designa (marca, apunta, referencia) un element d'una llista o un altre contenidor
- Operacions sobre iteradors:
  - Avançar al següent element: `++it`
  - Retrocedir a l'anterior: `--it`
  - Comparar iteradors: `it1==it2`, `it1!=it2`
  - Accedir a l'objecte designat: `*it`

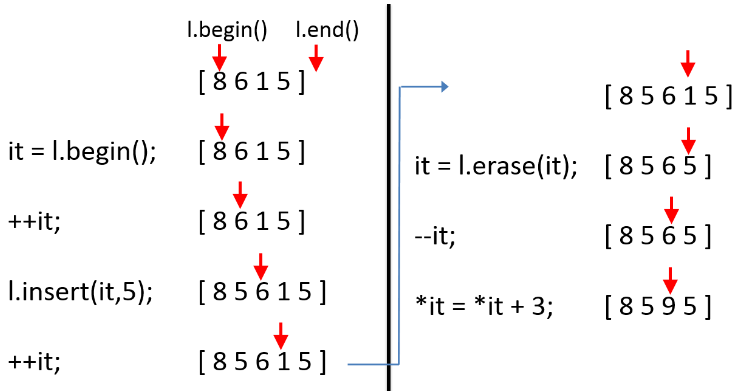
Iteradors

- Llistes i iteradors “treballen” coordinadament
- Operacions de llistes amb iteradors:
  - $L.\texttt{insert}(it, x)$: insereix a la llista L un nou element x com a predecessor de l’element apuntat per it
  - $it = L.\texttt{erase}(it)$: elimina de la llista L l’element apuntat per it; retorna un iterador al successor de l’element esborrat
- Hi ha altres versions d’`insert` i `erase` però recomanem fer servir només aquestes.

Iteradors

- Llistes i iteradors “treballen” coordinadament
- Operacions que ens tornen iteradors:
  - L.`begin()`: torna un iterador apuntant al primer element de la llista L
  - L.`end()`: torna un iterador apuntant “fora”—a un element fictici successor de l’últim—de la llista L
  - Si L és buida, aleshores `L.begin() == L.end()`

Exemple d'evolució d'una llista



Iteradors

```
list<Estudiant> l;
list<string> lp;

list<Estudiant>::iterator it = l.begin();
list<Estudiant>::iterator it2 = l.end();
list<string>::iterator it3 = lp.begin();

it = it3; // error!! són de tipus diferents
```

Cada tipus d'iterador es defineix com a subclasse de la classe "contenidora"

Iteradors: Recorreguts

Esquema freqüent:

```
list<T> L;
list<T>::iterator it = L.begin();
while (it != L.end() and not condició sobre *it)) {
   accedir a *it
   ++it;
}
```

Iteradors constants

- Iteradors constants (`const_iterator`): prohibeixen modificar l'objecte referenciat per l'iterador
- S'han d'utilitzar per a recòrrer una llista rebuda per referència constant

```
list<Estudiant>::const_iterator it, it2;
it = it2;  // OK, it és constant
++it;      // OK
v = *it;   // OK
*it = v+3; // error!!!
```

Iteradors constants

```
void imprimir_llista(const list<Estudiant>& L) {
  for(list<Estudiant>::const_iterator it = L.begin();
      it != L.end(); ++it)
    (*it).escriure();
}
// en comptes de (*it).escriure() podem posar
// it -> escriure();
```

Part I

1. Estructures lineals: Generalitats
2. El tipus pila (`stack`)
3. El tipus cua (`queue`)
4. Llistes
   - Llistes i Iteradors
   - Especificació de la classe Llista
   - Exemples d'operacions amb llistes
   - Splice
   - Fusió ordenada
   - Creació de llistes

Especificació de la classe `list`

```
template <class T> class list {
public:
// Subclasses de la classe llista
      class iterator { ... };
      class const_iterator { ... };

// Constructores

/* Pre: cert */
/* Post: El resultat es una llista sense cap element */
list();
```

Especificació de la classe genèrica `Llista`

```
// Modificadores
/* Pre: cert */
/* Post: La llista implícita queda buida */
void clear();

/* Pre: it referencia algun element existent a_i a la llista o
        és igual a end(), la llista és [a_1,...,a_n] */
/* Post: L'element s'ha inserit davant de l'element referenciat
         per it, la llista és ara [a_1,...,x,a_i,...] */
void insert(iterator it, const T& x);
```

Especificació de la classe genèrica `Llista`

```
/* Pre: it referencia algun element a_i existent a la
        llista [a_1,...,a_n], n > 0 */
/* Post: S'ha eliminat l'element referenciat per it, la llista
         és ara [a_1,...,a_-1, a_{i+1} ...,a_n] i torna
         un iterador al sucessor de l'element eliminat */
iterator erase(iterator it);

/* Pre: l = [y_1,...,y_m], l i la llista implícita són
        objectes diferents, i it referencia algún element x_i de
        la llista implícita [x_1,...,x_n] */
/* Post: La llista implícita és ara
        [x_1,...,x_{i-1}, y_1,...,y_m, x_i,...,x_n] i l és buida */
void splice(iterator it, list& l);
```

Especificació de la classe genèrica `Llista`

```
// Consultores

/* Pre: cert */
/* Post: torna cert si i només si la llista és buida */
bool empty() const;

/* Pre: cert */
/* Post: torna el nombre d'elements de la llista*/
int size() const;
```

Especificació de la classe genèrica `Llista`

```
    ...
// tornen iteradors al primer element de la llista
 const_iterator begin() const;
 iterator begin();
// tornen iteradors a l'element fictici succesor de l'últim
// de la llista
 const_iterator end() const;
 iterator end();
 private:
    ...
```

Errors freqüents en l’ús d’iteradors

Errors freqüents en l'ús d'iteradors

- Fer servir un iterador amb una llista a la qual no està vinculat.

Errors freqüents en l'ús d'iteradors

- Fer servir un iterador amb una llista a la qual no està vinculat.
- Tenir l'iterador posicionat al `begin()` i retrocedir.

Errors freqüents en l'ús d'iteradors

- Fer servir un iterador amb una llista a la qual no està vinculat.
- Tenir l'iterador posicionat al `begin()` i retrocedir.
- Tenir l'iterador posicionat a l'`end()` i avançar.

Errors freqüents en l'ús d'iteradors

- Fer servir un iterador amb una llista a la qual no està vinculat.
- Tenir l'iterador posicionat al `begin()` i retrocedir.
- Tenir l'iterador posicionat a l'`end()` i avançar.
- Tenir l'iterador posicionat a l'`end()` i consultar o modificar l'element.

Errors freqüents en l'ús d'iteradors

- Fer servir un iterador amb una llista a la qual no està vinculat.
- Tenir l'iterador posicionat al `begin()` i retrocedir.
- Tenir l'iterador posicionat a l'`end()` i avançar.
- Tenir l'iterador posicionat a l'`end()` i consultar o modificar l'element.
- Tenir l'iterador posicionat a l'`end()` i esborrar l'element.

Part I

1. Estructures lineals: Generalitats
2. El tipus pila (`stack`)
3. El tipus cua (`queue`)
4. Llistes
   - Llistes i Iteradors
   - Especificació de la classe Llista
   - Exemples d'operacions amb llistes
   - Splice
   - Fusió ordenada
   - Creació de llistes

Sumar tots els elements d'una llista d'enters

```
/* Pre: cert */
/* Post: El resultat és la suma dels elements de l */
int suma(const list<int>& l) {
    int s = 0;
    for (list<int>::const_iterator it = l.begin();
                                   it != l.end();
                                   ++it) {
        s += *it;
    }
    return s;
}
```

Cerca senzilla en una llista d'enters

```
/* Pre: cert */
/* Post: El resultat indica si x és o no a l */
bool pertany(const list<int>& l, int x) {
    list<int>::const_iterator it = l.begin();
    while (it != l.end() and (*it != x))
      ++it;
    return it != l.end();
}
```

Exercici: cerca en una llista d'estudiants

```
/* Pre: cert */
/* Post: El resultat ens indica si hi ha algun estudiant
   amb dni x a l o no */
bool pertany(const list<Estudiant>& l, int x);
```

Modificar una llista sumant un valor `k` a tots els elements

```
/* Pre: l = [x_1, ..., x_n] */
/* Post: l = [x_1 + k, x_2 + k, ..., x_n + k] */
void suma_k(list<int>& l, int k) {
    list<int>::iterator it = l.begin();
    while (it != l.end()) {
        *it += k;
        ++it;
    }
}
```

Una alternativa

En comptes de fer

```
*it += k;
++it;
```

podriem eliminar l'element i tornar a afegir-ho

```
int aux = (*it) + k;
it = l.erase(it); // it apunta al successor
l.insert(it,aux);
```

però és molt menys eficient (implica creació+destrucció d'objectes!)

Dir si una llista és capicua

[4,8,5,8,4], [7], [4,8,8,4] són capicues

```
/* Pre: cert */
/* Post: El resultat diu si l es capicua */
bool capicua(const list<int>& l);
```

Dir si una llista és capicua

```
bool capicua(const list<int>& l) {
   list<int>::const_iterator it1 = l.begin();
   list<int>::const_iterator it2 = l.end();
   for (int i = 0; i < l.size()/2; ++i) {
       --it2;
       if (*it1 != *it2) return false;
       ++it1;
   }
   return true;
}
```

Dir si una llista és capicua

- Exercici: Penseu com fer-ho sense usar `l.size()`.
- Cada element s'ha de consultar un cop com a molt.
- Recordeu que no es pot comparar `it1 < it2`

Inserint elements ordenadament

Exemple:
Donada una llista l d'strings en ordre alfabètic no decreixent i un nou string s, inserir s a la llista l, mantentint l'ordre.

```
// Pre: l = L
// Post: l conté els elements d'L i s, i està en ordre
// no decreixent
void inserir_ordenadament(list<string>& l, string s);
```

Inserint elements ordenadament

```
void inserir_ordenadament(list<string>& l, string s) {
  list<string>::iterator it = ...;
     ...
  // it == l.end() ó *it és un string ≥ s
  // per tant s s'ha d'inserir com a predecessor
  // de l'element apuntat per it
  l.insert(it, s);
}
```

Inserint elements ordenadament

```
void inserir_ordenadament(list<string>& l, string s) {
  list<string>::iterator it = l.begin();
  while (it != l.end() and *it < s) ++it;
  // it == l.end() ó *it és un string ≥ s
  // per tant s s'ha d'inserir com a predecessor
  // de l'element apuntat per it
  l.insert(it, s);
}
```

Part I

1. Estructures lineals: Generalitats
2. El tipus pila (`stack`)
3. El tipus cua (`queue`)
4. Llistes
   - Llistes i Iteradors
   - Especificació de la classe Llista
   - Exemples d'operacions amb llistes
   - Splice
   - Fusió ordenada
   - Creació de llistes

Splice: Insert a l'engrós!

- Si `l1 = [1,2,3,4,5,6]`, `it` apunta al 4, i `l2 = [10,20,30]` llavors

  `l1.splice(it,l2),`

  queda

Splice: Insert a l'engrós!

- Si `l1 = [1,2,3,4,5,6]`, `it` apunta al 4, i `l2 = [10,20,30]` llavors

  `l1.splice(it,l2),`

  queda
  `l1 = [1,2,3,10,20,30,4,5,6]`, `it` apunta a 4, `l2` buida

Splice: Insert a l’engrós!

- Si `l1 = [1,2,3,4,5,6]`, `it` apunta al 4, i `l2 = [10,20,30]` llavors

  ```
  l1.splice(it,l2),
  ```

  queda
  `l1 = [1,2,3,10,20,30,4,5,6]`, `it` apunta a 4, `l2` buida
- Per concatenar dues llistes farem:
  `l1.splice(l1.end(),l2)`
- La STL de C++ té variants més complexes de `splice` que fan altres tipus de “transferència” de continguts entre llistes
- El cost de `splice` és constant, no depèn de les longituds de les llistes

Part I

1. Estructures lineals: Generalitats
2. El tipus pila (`stack`)
3. El tipus cua (`queue`)
4. Llistes
   - Llistes i Iteradors
   - Especificació de la classe Llista
   - Exemples d'operacions amb llistes
   - Splice
   - Fusió ordenada
   - Creació de llistes

Fusió ordenada de llistes

Suposem que tenim una llista l d'Estudiants ordenada per DNI i sense repeticions, i una altra llista també ordenada per DNI, sense repeticions, d'elements del tipus

```
struct Actualitzacio {
  char op; // 'a' = alta, 'b' = baixa
           // 'm' = modificació
  Estudiant est;
};
```

En aquesta segona llista es detalla: estudiants que s'han de donar d'alta (NO estan a la llista l); estudiants que s'han de donar de baixa (SÍ estan a la llista l) i estudiants als quals se'ls ha de modificar o agregar nota (SÍ estan a la llista l).

Fusió ordenada de llistes

```
/* Pre: l = L, l és una llista d'Estudiant ordenada ascendentment
        sense DNI repetits, lact = A, lact és una llista
        d'actualitzacions ordenada ascendentment per DNI
        sense DNI repetits. Si un element d'A conte 'b' o 'm',
        el DNI del seu Estudiant associat està a L, si un element
        d'A conte 'a', el DNI del seu Estudiant associat no hi és
        a L */
/* Post: l és L després d'aplicar les modificacions d'A */
void actualitza_llista_Estudiants(list<Estudiant>& l,
                      const list<Actualitzacio>& lact);
```

Una solució eficient d'aquest problema ha d'aprofitar que les dues llistes estàn ordenades alfabèticament!

Fusió ordenada de llistes

Per exemple (substituint DNIs per noms per fer-ho més entenedor) si

$$l = [\langle \text{ALICE}, 3.5\rangle, \langle \text{BOB}, 4.1\rangle, \langle \text{CHARLIE}, 7.4\rangle, \langle \text{DAISY}, 6.8\rangle, \langle \text{HELEN}, 9.1\rangle, \langle \text{JOHN}, 3.7\rangle, \langle \text{MARY}, 5.3\rangle]$$

i la llista d'actualitzacions és

$$lact = [\langle \text{'a'}, \langle \text{ALBERT}, \text{NP}\rangle\rangle, \langle \text{'b'}, \langle \text{BOB}, \ldots\rangle\rangle, \langle \text{'m'}, \langle \text{HELEN}, 10\rangle\rangle, \langle \text{'m'}, \langle \text{JOHN}, 4.1\rangle\rangle, \langle \text{'a'}, \langle \text{PETER}, 5.5\rangle\rangle]$$

el resultat d'actualitzar l seria

$$l = [\langle \text{ALBERT}, \text{NP}\rangle, \langle \text{ALICE}, 3.5\rangle, \langle \text{CHARLIE}, 7.4\rangle, \langle \text{DAISY}, 6.8\rangle, \langle \text{HELEN}, 10\rangle, \langle \text{JOHN}, 4.1\rangle, \langle \text{MARY}, 5.3\rangle, \langle \text{PETER}, 5.5\rangle]$$

Fusió ordenada de llistes

```
void actualitza_llista_Estudiants(list<Estudiant>& l,
                      const list<Actualitzacio>& lact) {
  list<Estudiant>::iterator it = l.begin();
  list<Actualitzacio>::const_iterator itact = lact.begin();
  while (it != l.end() and itact != lact.end()) {
    if (it -> consultar_DNI() <
        itact -> est.consultar_DNI())
     // l'estudiant al que apunta it no està afectat
     // per cap actualització
        ++it;
    ...
  }
  // processar la resta d'actualitzacions
  // que pugui haver
  ...
}
```

Fusió ordenada de llistes

```
...
else if (it -> consultar_DNI() >
            itact -> est.consultar_DNI()) {
 // això ha de ser un 'alta' necessàriament
 l.insert(it, itact -> est);
 ++itact;
} else {
 // it -> consultar_DNI() == itact -> est.consultar_DNI()
 // això és necessàriament una 'baixa'
 // o una 'modificació'
 if (itact -> op == 'b')
   it = l.erase(it);
 else
   {*it = itact -> est; ++it}
 ++itact;
}
```

Fusió ordenada de llistes

- Si el bucle acaba perquè `it == l.end()` poden quedar actualitzacions per fer (en cas que `itact != lact.end()`), que només poden ser altes. Cal afegir un segon bucle:

Fusió ordenada de llistes

- Si el bucle acaba perquè `it == l.end()` poden quedar actualitzacions per fer (en cas que `itact != lact.end()`), que només poden ser altes. Cal afegir un segon bucle:

```
while (itact != lact.end()){
        l.insert(it,itact->est); ++itact;
}
```

Fusió ordenada de llistes

- Si el bucle acaba perquè `it == l.end()` poden quedar actualitzacions per fer (en cas que `itact != lact.end()`), que només poden ser altes. Cal afegir un segon bucle:

```
while (itact != lact.end()){
        l.insert(it,itact->est); ++itact;
}
```

`it` serà `l.end()` durant l'execució d'aquest bucle.

Fusió ordenada de llistes

- Si el bucle acaba perqué `it == l.end()` poden quedar actualitzacions per fer (en cas que `itact != lact.end()`), que només poden ser altes. Cal afegir un segon bucle:

```
while (itact != lact.end()){
        l.insert(it,itact->est); ++itact;
}
```

  `it` serà `l.end()` durant l'execució d'aquest bucle.

- Si el bucle acaba perqué `itact == lact.end()` en aquest exemple no cal fer res més perqué com no s'han de fer més actualitzacions `l` no canvia.

Fusió ordenada de llistes

- Si el bucle acaba perquè `it == l.end()` poden quedar actualitzacions per fer (en cas que `itact != lact.end()`), que només poden ser altes. Cal afegir un segon bucle:

```
while (itact != lact.end()){
        l.insert(it,itact->est); ++itact;
}
```

  `it` serà `l.end()` durant l'execució d'aquest bucle.

- Si el bucle acaba perquè `itact == lact.end()` en aquest exemple no cal fer res més perquè com no s'han de fer més actualitzacions `l` no canvia.
- En general cal un tractament final per cada llista per tractar la part no tractada en el bucle principal. En aquest exemple només ens cal per una de les dues llistes.

Fusió ordenada de llistes

Suposem que l té $n = 10000$ elements i que $lact$ conté $m = 1000$ actualitzacions. Si l'actualització de l la fessim amb

```
itact = lact.begin();
while (itact != lact.end()) {
  Estudiant e = itact -> est;
  char op = itact -> op;
  if (op == 'a')
     afegeix(l, est);
  else if (op == 'b')
     elimina(l, est);
  else
     modifica(l, est);
}
```

Fusió ordenada de llistes

o quelcom equivalent hauriem de fer unes $n \cdot m = 10^7$ (deu millions) d'operacions en el pitjor dels casos i de l'ordre de 5 millions d'operacions en promig, ja que operacions com `afegir`, `elimina` o `modifica` han de recòrrer la llista l (en promig la meitat).

Fusió ordenada de llistes

Però el nostre algorisme de “fusió ordenada” avança en cada pas/iteració en una de les dues llistes (o totes dues) i per tant el nombre d’operacions que farem serà de l’ordre d’$n + m = 11000$ operacions. La diferència és enorme! És pràcticament 1000 vegades més eficient!

Part I

1. Estructures lineals: Generalitats
2. El tipus pila (`stack`)
3. El tipus cua (`queue`)
4. Llistes
   - Llistes i Iteradors
   - Especificació de la classe Llista
   - Exemples d'operacions amb llistes
   - Splice
   - Fusió ordenada
   - Creació de llistes

Funció

Si volem conservar la llista original i que la llista modificada sigui nova, podem fer una funció, on haurem de fer servir l'operació `insert` per generar la llista resultat.

```
list<int> suma_llista_k(const list<int>& l, int k)
/* Pre: cert */
/* Post: Cada element del resultat és la suma de k
                  i l'element de l a la seva mateixa posició */
{
  list<int>::const_iterator it=l.begin();
  list<int> l2;
  list<int>::iterator it2=l2.begin();
  while (it != l.end()) {
    l2.insert(it2,*it+k);
    ++it;
  }
  return l2;
}
```

Acció

Si obtenim la nova llista amb una acció, ens estalviarem l'assignació al resultat quan fem la crida corresponent.

```
void suma_llista_k(const list<int>& l, list<int> &l2, int k)
/* Pre: l2 és buida */
/* Post: Cada element de l2 és la suma de k i l'element de l a
         la seva mateixa posició */
{
  list<int>::const_iterator it=l.begin();
  list<int>::iterator it2=l2.begin();
  while (it != l.end()) {
    l2.insert(it2,*it+k);
    ++it;
  }
}
```