
Esquivar a un altre iterador a base de moure's un pas mésX82913_ca

Típicament, executar ++ sobre un iterador que es troba a l'end de la llista produeix error d'execució, i executar -- sobre un iterador que es troba al begin de la llista també produeix error d'execució. Per començar, en aquest exercici modificarem la subclasse `iterator` de la classe `List` de manera que els errors d'execució abans esmentats ja no es produiran. Simplement, en tals casos els iteradors no es mouran.

Després modificarem la classe `iterator` afegint dos nous mètodes `dodge` i `stopDodge`, i canviant el comportament dels mètodes ++ i -- com descrivim a continuació.

El nou mètode `dodge` rebrà un altre `iterator` com a paràmetre (és a dir, un iterador del mateix tipus, tot i que potser apunta a un element d'una llista diferent). Una crida `it0.dodge(it1)` provocarà que, a partir d'ara, `it0` intenti evitar apuntar al mateix lloc que `it1`, a base de prolongar amb un pas més els moviments que ho poden provocar.

Més concretament, suposem que `it0` no apunta a l'end de la llista. Llavors, amb una crida `it0++` o `++it0`, l'iterador `it0` farà un pas cap a l'end de la llista. Si després d'aquest moviment encara no ha arribat a l'end i passa que `it0` apunta al mateix lloc que `it1`, llavors encara farà un pas més cap a l'end de la llista.

Anàlogament, suposem que `it0` no apunta al begin de la llista. Llavors, amb una crida `it0--` o `--it0`, l'iterador `it0` farà un pas cap al begin de la llista. Si després d'aquest moviment encara no ha arribat al begin i passa que `it0` apunta al mateix lloc que `it1`, llavors encara farà un pas més cap al begin de la llista.

En altres paraules, `it0` mira de fer dos passos si fent-ne només un queda apuntant al mateix lloc que `it1`.

Fixeu-vos que la crida `it0.dodge(it1)` no imposa restriccions al moviment de `it1`. Per tant, aquesta crida no implica que `it1` intenti esquivar `it0`.

Una crida posterior `it0.dodge(it2)` posa restriccions al moviment de `it0` respecte de `it2`, però també deixa sense efecte la crida anterior `it0.dodge(it1)`, és a dir, cancel·la les restriccions del moviment de `it0` respecte de `it1`.

Una crida posterior `it0.stopDodge()` cancel·la les restriccions del moviment de `it0` respecte de qualsevol altre iterador.

Fixeu-vos en aquest exemple per tal d'acabar d'entendre-ho:

```
List<int> l0, l1;
List<int>::iterator a, b, c, d;

l0.push_back(1); // l0: 1,
l0.push_back(2); // l0: 1,2,
l0.push_back(3); // l0: 1,2,3,
l1.push_back(4); // l1: 4,
l1.push_back(5); // l1: 4,5,
l1.push_back(6); // l1: 4,5,6,

a = l0.begin(); // l0: 1a,2,3,
b = l0.end(); // l0: 1a,2,3,b
c = l1.begin(); // l1: 4c,5,6,
d = l1.end(); // l1: 4c,5,6,d
```

```

a--;          // 10: 1a,2,3,b
a++;          // 10: 1,2a,3,b
b++;          // 10: 1,2a,3,b
b--;          // 10: 1,2a,3b,
a.dodge(b);
a++;          // 10: 1,2,3b,a
a--;          // 10: 1,2a,3b,
b--;          // 10: 1,2ab,3,
a++;          // 10: 1,2b,3a,
a--;          // 10: 1a,2b,3,
b--;          // 10: 1ab,2,3,
b--;          // 10: 1ab,2,3,
a++;          // 10: 1b,2a,3,
a--;          // 10: 1ab,2,3,
b++;          // 10: 1a,2b,3,
a++;          // 10: 1,2b,3a,
b++;          // 10: 1,2,3ab,
b++;          // 10: 1,2,3a,b
a++;          // 10: 1,2,3,ab
a++;          // 10: 1,2,3,ab
b++;          // 10: 1,2,3,ab
a.dodge(c);
c.dodge(d);
d.dodge(c);
b--;          // 10: 1,2,3b,a  l1: 4c,5,6,d
a--;          // 10: 1,2,3ab,  l1: 4c,5,6,d
c--;          // 10: 1,2,3ab,  l1: 4c,5,6,d
c++;          // 10: 1,2,3ab,  l1: 4,5c,6,d
c++;          // 10: 1,2,3ab,  l1: 4,5,6c,d
c++;          // 10: 1,2,3ab,  l1: 4,5,6,cd
d--;          // 10: 1,2,3ab,  l1: 4,5,6d,c
c--;          // 10: 1,2,3ab,  l1: 4,5c,6d,
d--;          // 10: 1,2,3ab,  l1: 4d,5c,6,
c--;          // 10: 1,2,3ab,  l1: 4cd,5,6,
d--;          // 10: 1,2,3ab,  l1: 4cd,5,6,
c--;          // 10: 1,2,3ab,  l1: 4cd,5,6,
d++;          // 10: 1,2,3ab,  l1: 4c,5d,6,
c++;          // 10: 1,2,3ab,  l1: 4,5d,6c,
d++;          // 10: 1,2,3ab,  l1: 4,5,6c,d
c++;          // 10: 1,2,3ab,  l1: 4,5,6,cd
d++;          // 10: 1,2,3ab,  l1: 4,5,6,cd
c++;          // 10: 1,2,3ab,  l1: 4,5,6,cd
c.stopDodge();
c--;          // 10: 1,2,3ab,  l1: 4,5,6c,d
d--;          // 10: 1,2,3ab,  l1: 4,5d,6c,
c--;          // 10: 1,2,3ab,  l1: 4,5dc,6,
c--;          // 10: 1,2,3ab,  l1: 4c,5d,6,
c++;          // 10: 1,2,3ab,  l1: 4,5cd,6,

```

```

c++;                // 10: 1,2,3ab, 11: 4,5d,6c,
d.stopDodge();
d++;                // 10: 1,2,3ab, 11: 4,5,6cd,

```

D'entre els fitxers que s'adjunten en aquest exercici, trobareu `list.hh`, a on hi ha una implementació de la classe genèrica `List`. Haureu d'implementar els dos nous mètodes `dodge` i `stopDodge` dins `list.hh` a la part pública de la classe `iterator` (podeu trobar les capçaleres comentades dins `list.hh`), i modificar els dos mètodes `++` i els dos mètodes `--` convenientment (en realitat només cal modificar el pre-increment i el pre-decrement perquè el post-increment i post-decrement criden als primers). Necessitareu també algun atribut addicional per tal de recordar si l'iterador té un `dodge` actiu i amb qui, amb les convenient inicialitzacions.

Més concretament, heu de fer els canvis que s'indiquen en algunes parts del codi de `list.hh`:

```

// Iterators mutables
class iterator {
    friend class List;
private:
    List *plist;
    Item *pitem;
    // Add new attributes to remember if the iterator has an active 'dodge'
    // and with which other iterator.

public:

    iterator() {
        // Add initialization of new attributes.
    }

    // Adapt this function so that moving beyond boundaries does not trigger error
    // but leaves the iterator unchanged instead.
    // Also, add the necessary adaptations so that, the method attempts one extra
    // when there is an active 'dodge' and the first move implies pointing to the
    // the other involved iterator.
    // Preincrement
    iterator operator++()
    /* Pre: el p.i apunta a un element E de la llista,
       que no és el end() */
    /* Post: el p.i apunta a l'element següent a E
       el resultat és el p.i. */
    {
        if (pitem == &(plist->itemsup)) {
            cerr << "Error: ++iterator at the end of list" << endl;
            exit(1);
        }
        pitem = pitem->next;
        return *this;
    }
}

```

...

```
// Adapt this function so that moving beyond boundaries does not trigger er
// but leaves the iterator unchanged instead.
// Also, add the necessary adaptations so that, the method attempts one ext
// when there is an active 'dodge' and the first move implies pointing to t
// the other involved iterator.
// Predecrement
iterator operator--()
/* Pre: el p.i apunta a un element E de la llista que
   no és el begin() */
/* Post: el p.i apunta a l'element anterior a E,
   el resultat és el p.i. */
{
    if (pitem == plist->iteminf.next) {
        cerr << "Error: --iterator at the beginning of list" << endl;
        exit(1);
    }
    pitem = pitem->prev;
    return *this;
}
```

...

```
// Pre: 'it' != 'this'
// Post: Once executed, any move attempt (++ or --) on 'this' will cause be
//        extended with one extra move attempt if the first move makes 'this
//        point to the same place as 'it'.
//        All former dodge's are cancelled.
// Remove comment marks and implement this function:
// void dodge(iterator &it) {
// }

// Pre: 'this' has an active dodge.
// Post: All former dodge's are cancelled.
// Remove comment marks and implement this function:
// void stopDodge() {
// }
```

...

No cal decidir que passa amb assignacions entre iteradors existents, doncs no es consideraran en els jocs de proves.

D'entre els fitxers que s'adjunten a l'exercici també hi ha `main.cc` (programa principal), i el podeu compilar directament, doncs inclou `list.hh`. Només cal que pugueu `list.hh` al jutge.

Entrada

L'entrada del programa comença amb una declaració d'unes quantes llistes (10, 11, ...) i uns quants iteradors (a, b, c, ...), i després té una seqüència de comandes sobre les llistes i els iteradors declarats. Com que ja us oferim el `main.cc`, no cal que us preocupeu d'implementar la lectura d'aquestes entrades. Només cal que implementeu la extensió de la classe `iterator` abans esmentada.

Per simplificar, no hi haurà comandes que eliminin elements de les llistes, com `pop_back`, `pop_front` i `erase`. Podeu suposar que les comandes no fan coses estranyes, com fer que un iterador tingui un `dodge` a si mateix, i que sempre que un iterador sigui mogut, aquest estarà apuntant a alguna posició d'alguna llista. Podeu suposar que les comandes faran `stopDodge` només sobre iteradors que tinguin un `dodge` actiu. Però pot ser el cas que es faci un `dodge` sobre un iterador que ja tingui un `dodge` actiu. Com mencionavem abans, en aquestes situacions només l'últim `dodge` aplica.

Sortida

Per a cada comanda d'escriptura sobre la sortida s'escriurà el resultat corresponent. El `main.cc` que us oferim ja fa això. Només cal que implementeu la extensió de la classe `iterator` abans esmentada.

Exemple d'entrada 1

```
List<int> l0 , l1 ;
List<int>::iterator a , b , c , d ;

l0 .push_back( 1 ); // l0: 1,
l0 .push_back( 2 ); // l0: 1,2,
l0 .push_back( 3 ); // l0: 1,2,3,
l1 .push_back( 4 ); // l1: 4,
l1 .push_back( 5 ); // l1: 4,5,
l1 .push_back( 6 ); // l1: 4,5,6,

a = l0 .begin(); // l0: 1a,2,3,
b = l0 .end(); // l0: 1a,2,3,b
c = l1 .begin(); // l1: 4c,5,6,
d = l1 .end(); // l1: 4c,5,6,d

cout<< l0 <<endl;
cout<< l1 <<endl;

a --; // l0: 1a,2,3,b

cout<< l0 <<endl;
cout<< l1 <<endl;

a ++; // l0: 1,2a,3,b

cout<< l0 <<endl;
cout<< l1 <<endl;

b ++; // l0: 1,2a,3,b

cout<< l0 <<endl;
cout<< l1 <<endl;

b --; // l0: 1,2a,3b,

cout<< l0 <<endl;
cout<< l1 <<endl;

a .dodge( b );
a ++; // l0: 1,2,3b,a

cout<< l0 <<endl;
cout<< l1 <<endl;

a --; // l0: 1,2a,3b,

cout<< l0 <<endl;
cout<< l1 <<endl;

b --; // l0: 1,2ab,3,

cout<< l0 <<endl;
cout<< l1 <<endl;

a ++; // l0: 1,2b,3a,

cout<< l0 <<endl;
cout<< l1 <<endl;

a --; // l0: 1a,2b,3,

cout<< l0 <<endl;
cout<< l1 <<endl;

b --; // l0: 1ab,2,3,

cout<< l0 <<endl;
cout<< l1 <<endl;
```


Exemple de sortida 1

```
d ++; // 10: 1,2,3ab, 11: 4,5,6c,d
1a,2,3,b
4c,5,6,d
cout<< 10 <<endl; 1a,2,3,b
cout<< 11 <<endl; 4c,5,6,d

c ++; // 10: 1,2,3ab, 11: 1,2,3,b
4c,5,6,d
1,2a,3,b
cout<< 10 <<endl; 4c,5,6,d
cout<< 11 <<endl; 1,2a,3b,

d ++; // 10: 1,2,3ab, 11: 4c,5,6,d
1,2,3b,a
4c,5,6,d
cout<< 10 <<endl; 1,2a,3b,
cout<< 11 <<endl; 4c,5,6,d

c ++; // 10: 1,2,3ab, 11: 1,2,3a,b
4c,5,6,d
1,2b,3a,
cout<< 10 <<endl; 4c,5,6,d
cout<< 11 <<endl; 1a,2b,3,
4c,5,6,d

c .stopDodge();
c --; // 10: 1,2,3ab, 11: 1a,b,2,3,
4c,5,6,d
1ab,2,3,
cout<< 10 <<endl; 4c,5,6,d
cout<< 11 <<endl; 1b,2a,3,

d --; // 10: 1,2,3ab, 11: 4c,5,6,d
1ab,2,3,
4c,5,6,d
cout<< 10 <<endl; 1a,2b,3,
cout<< 11 <<endl; 4c,5,6,d

c --; // 10: 1,2,3ab, 11: 1,2,3ab,
4c,5,6,d
1,2,3ab,
cout<< 10 <<endl; 4c,5,6,d
cout<< 11 <<endl; 1,2,3a,b

c --; // 10: 1,2,3ab, 11: 4c,5,6,d
1,2,3,ab
4c,5,6,d
cout<< 10 <<endl; 1,2,3,ab
cout<< 11 <<endl; 4c,5,6,d

c ++; // 10: 1,2,3ab, 11: 1,2,3b,a
4c,5,6,d
1,2,3ab,
cout<< 10 <<endl; 4c,5,6,d
cout<< 11 <<endl; 1,2,3ab,

c ++; // 10: 1,2,3ab, 11: 4c,5,6,d
1,2,3ab,
4c,5,6,d
1,2,3ab,
cout<< 10 <<endl; 1,2,3ab,
cout<< 11 <<endl; 4,5c,6,d

d .stopDodge();
1,2,3ab,
4,5,6c,d
1,2,3ab,
4,5,6,cd
1,2,3ab,
cout<< 10 <<endl; 4,5,6c,d
cout<< 11 <<endl; 1,2,3ab,
4,5c,6d,
1,2,3ab,
```

```

4d,5c,6,
1,2,3ab,
4cd,5,6,
1,2,3ab,
4cd,5,6,
1,2,3ab,
4cd,5,6,
1,2,3ab,
4c,5d,6,
1,2,3ab,
4,5d,6c,
1,2,3ab,
4,5,6c,d
1,2,3ab,
4,5,6,cd
1,2,3ab,
4,5,6,cd

```

Exemple d'entrada 2

```

List<int> l0 , l1 ;
List<int>::iterator a , b , c , d , e ;
a = l1 .begin();
b = l0 .begin();
c = l1 .begin();
d = l1 .begin();
e = l1 .begin();
cout<< l0 <<endl;
e --;
cout<< l0 <<endl;
d ++;
c .dodge( d );
cout<< l0 .size()<<endl;
cout<< l1 <<endl;
++ a ;
l1 .push_back( 1 );
l1 .insert( c , -1 );
cout<< l1 <<endl;
l0 .push_back( 0 );
l1 .push_front( -4 );
cout<< l1 <<endl;
cout<< l1 <<endl;
e --;
cout<< l0 <<endl;
++ a ;
l0 .push_front( 0 );
l1 .push_back( -4 );
c .dodge( d );
-- b ;
e --;
e --;
d .dodge( b );
d ++;
-- e ;
e ++;
cout<< l0 <<endl;
cout<< l0 .size()<<endl;
++ b ;
cout<< l0 <<endl;
-- d ;
c .dodge( e );

```

```

1,2,3ab,
4,5,6,cd
1,2,3ab,
4,5,6c,d
1,2,3ab,
4,5d,6c,
1,2,3ab,
4,5cd,6,
1,2,3ab,
4c,5d,6,
1,2,3ab,
4,5cd,6,
1,2,3ab,
4,5d,6c,
1,2,3ab,
4,5d,6c,

```

```

++ c ;
b .dodge( d );
cout<< l1 <<endl;
e .dodge( a );
-- b ;
cout<< l1 <<endl;
e .dodge( d );
-- b ;
++ a ;
e .dodge( a );
cout<< l0 <<endl;
cout<< l1 <<endl;
a ++;
cout<< l0 <<endl;
cout<< l0 <<endl;
d --;
cout<< l1 .size()<<endl;
cout<< l1 <<endl;
++ e ;
d ++;
e = l0 .begin();
cout<< l0 .size()<<endl;
cout<< l1 .size()<<endl;
a --;
l0 .insert( b , 4 );
a --;
-- d ;
a --;
d .dodge( c );
l1 .push_front( -2 );
cout<< l1 <<endl;
cout<< l1 .size()<<endl;
b .dodge( c );
d ++;
c --;
l1 .insert( d , 2 );
cout<< l0 <<endl;
cout<< l0 <<endl;
cout<< l0 <<endl;
cout<<* a <<endl;
l0 .push_back( 3 );
c ++;

```



```

c --;
cout<<* b <<endl;
cout<< 10 <<endl;
cout<< l1 .size()<<endl;
d = 10 .begin();
++ b ;
cout<< l1 <<endl;
d --;
cout<< 10 <<endl;
cout<<* c <<endl;
d ++;
10 .insert( e , 2 );
e ++;
e --;
-- b ;
a = l1 .end();
-- e ;
-- e ;
a --;
e ++;
10 .insert( b , 2 );
cout<< l1 <<endl;
b --;
c ++;
l1 .push_back( 1 );
c = l1 .begin();
b --;
l1 .push_back( 4 );
cout<< l1 <<endl;
l1 .push_front( 3 );
l1 .insert( c , -2 );
c ++;
a ++;
b = l1 .end();
e ++;
cout<< 10 <<endl;
cout<< l1 <<endl;
e ++;
cout<< l1 <<endl;
l1 .insert( a , -1 );
e = l1 .begin();
++ b ;
e .dodge( d );
l1 .push_back( 4 );
cout<<* a <<endl;
c .dodge( b );
10 .insert( d , 0 );
c = 10 .end();
cout<< l1 .size()<<endl;
b --;
c ++;
c ++;
c = 10 .end();
d .dodge( e );
++ d ;
d .stopDodge();
a .dodge( d );
cout<< 10 <<endl;
e .dodge( a );
cout<<* d <<endl;
cout<< l1 <<endl;

```

```

d .dodge( a );
cout<< l1 <<endl;
b = l1 .begin();
c --;
e .dodge( b );
++ e ;
cout<<* d <<endl;
++ c ;
d --;
++ d ;
cout<< 10 .size()<<endl;
b --;
cout<< 10 <<endl;
cout<< l1 .size()<<endl;
cout<< l1 <<endl;
-- e ;
l1 .push_back( 2 );
10 .insert( d , 1 );
e --;
a --;
a ++;
cout<< l1 <<endl;
c .dodge( e );
-- a ;
b .dodge( d );
cout<< l1 <<endl;
c --;
++ b ;
10 .push_front( -4 );
10 .insert( c , 0 );
b .dodge( e );
b ++;
10 .push_back( -3 );
a .stopDodge();
-- a ;
c ++;
cout<< l1 <<endl;
cout<< 10 <<endl;
cout<< l1 <<endl;
e --;
l1 .push_back( 1 );
l1 .push_back( 3 );
cout<<* b <<endl;
e .dodge( a );
cout<< 10 .size()<<endl;
cout<< l1 <<endl;
cout<< l1 <<endl;
a .dodge( e );
d .dodge( a );
-- e ;
++ c ;
cout<< 10 <<endl;
++ d ;
-- e ;
d ++;
-- c ;
cout<< 10 <<endl;
-- d ;
c ++;
-- c ;
e .dodge( d );

```

```

b .dodge( a );
c ++;
cout<< 10 <<endl;
cout<< 11 <<endl;

```

Exemple de sortida 2

```

b
b
0
acde
1, -1, acde
-4, 1, -1, acde
-4, 1, -1, acde
0, b
0, 0b,
2
0, 0, b
-4, 1e, -1, -4d, ac
-4, 1e, -1, -4d, ac
0b, 0,
-4, 1e, -1, -4d, ac
0b, 0,
0b, 0,
4
-4, 1e, -1d, -4, ac
2
4
-2, -4, 1a, -1d, -4, c
5
4, 0be, 0,
4, 0be, 0,
4, 0be, 0,
1
0
4, 0be, 0, 3,
6
-2, -4, 1a, -1, 2, -4c,
4d, 0e, 0b, 3,
-4
-2, -4, 1, -1, 2, -4ac,
-2c, -4, 1, -1, 2, -4a, 1, 4,
4, 2, 2e, 0d, 0, 3,
3, -2, -2, -4c, 1, -1, 2, -4, 1a, 4, b
3, -2, -2, -4c, 1, -1, 2, -4, 1a, 4, b
1
12
4, 2, 2, 0, 0, 0d, 3, c
0
3e, -2, -2, -4, 1, -1, 2, -4, -1, 1a, 4, 4b,
3e, -2, -2, -4, 1, -1, 2, -4, -1, 1a, 4, 4b,
0
7
4, 2, 2, 0, 0, 0d, 3, c
12
3b, -2e, -2, -4, 1, -1, 2, -4, -1, 1a, 4, 4,
3be, -2, -2, -4, 1, -1, 2, -4, -1, 1a, 4, 4, 2,
3be, -2, -2, -4, 1, -1, 2, -4, -1a, 1, 4, 4, 2,
3e, -2, -2b, -4, 1, -1, 2, -4a, -1, 1, 4, 4, 2,
-4, 4, 2, 2, 0, 0, 1, 0d, 0, 3, -3c,
3e, -2, -2b, -4, 1, -1, 2, -4a, -1, 1, 4, 4, 2,
-2
11
3e, -2, -2b, -4, 1, -1, 2, -4a, -1, 1, 4, 4, 2, 1, 3,
3e, -2, -2b, -4, 1, -1, 2, -4a, -1, 1, 4, 4, 2, 1, 3,
-4, 4, 2, 2, 0, 0, 1, 0d, 0, 3, -3, c

```

$-4, 4, 2, 2, 0, 0, 1, 0, 0, 3d, -3c,$
 $-4, 4, 2, 2, 0, 0, 1, 0, 0d, 3, -3, c$

$3e, -2, -2b, -4, 1, -1, 2, -4a, -1, 1, 4, 4, 2, 1, 3,$

Observació

Avaluació sobre 10 punts:

- Solució lenta: 5 punts.
- solució ràpida: 10 punts.

Entenem com a solució ràpida una que és correcta, on totes les operacions tenen cost constant (excepte l'escriptura de tota la llista per la sortida, que té cost lineal), i capaç de superar els jocs de proves públics i privats. Entenem com a solució lenta una que no és ràpida, però és correcta i capaç de superar els jocs de proves públics.

Informació del problema

Autor : PRO2

Generació : 2024-04-24 22:28:20

© *Jutge.org*, 2006–2024.

<https://jutge.org>