lkptr.h: Programación por Referencia Java para C++

Universidad de Costa Rica
Escuela de Ciencias de la
Computación e Informática

Prof. Adolfo Di Mare

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

adolfo.dimare@ecci.ucr.ac.cr

Agenda de la exposición

- Programación con punteros inteligentes
- Implementación de la clase "`lkptr<X>`"
- Construcción de una clase con punteros inteligentes
- Uso de referencias en programas C++
- El árbol binario implementado con referencias
- Peligro de usar referencias cíclicas
- Almacenamiento en contenedores STL
- Control de concurrencia
- Conclusión

- JavaKiller ???

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

```
{    auto_ptr<BigMatrix> p;
     auto_ptr<BigMatrix> q( new BigMatrix(50*1000,100*1000) );
     // ...
     q->foo( 15*1000 ); // "q" funciona como un puntero
     p = q; // Ahora "q" es NULL y "p" es el nuevo dueño
     // ...
}    // Aquí el destructor de "p" devuelve la memoria
```

```
template <class T> // OJO: "q" no es un argumento "const"
auto_ptr<T>& auto_ptr<T>::operator= ( auto_ptr<T>& q ) {
     if ( this != &q ) {
          delete m_ptr; // "m_ptr" referencia el objeto
          m_ptr = q.m_ptr; // nuevo dueño
          q.m_ptr = NULL; // deja a "q" en NULL
     }
     return *this;
}
```

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

Cualidades de la clase `lkptr<X>`

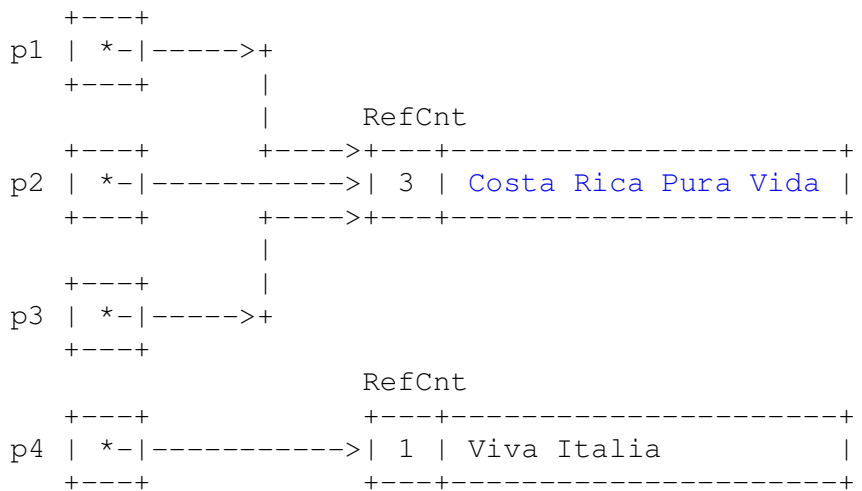
- No usa memoria dinámica adicional.
- No es "intrusiva" porque no requiere de un contador almacenado en el objeto referenciado.
- Es viable usar contenedores STL para almacenar referencias inteligentes "`lkptr<X>`".
- Es muy simple de usar pues la implementación completa está en el archivo de encabezado "`lkptr.h`".
- Es adecuada para aplicaciones en que se usa programación concurrente mediante múltiples hilos o procesos.

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la
Computación e Informática

Cuentas de referencia



adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

Objeto compartido con cuentas de referencia

```
lkptr<Chunche> foo() {
     lkptr<Chunche> p1, p2, p3;
     p3.reset( new Chunche( "Costa Rica Pura Vida" ) );
     p1 = p2 = p3; // los 3 comparte la instancia
     {    lkptr<Chunche> p4 (new Chunche( "Viva Italia" ) );
          // ...
     } // aquí "p4" es destruido

     return p2; // p1, p2, p3 son destruidos
}    //         ... el objeto referenciado persiste
```

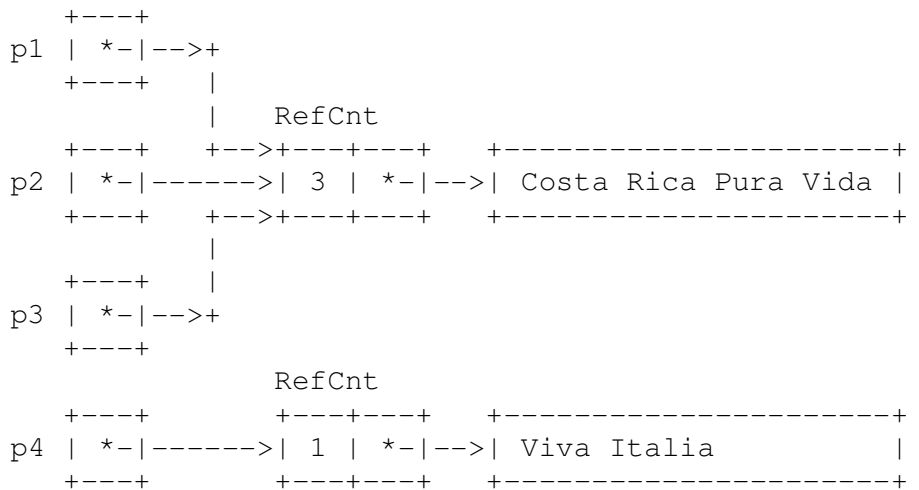
adolfo.dimare@ecci.ucr.ac.cr

http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la
Computación e Informática

Cuentas de referencia
Afuera

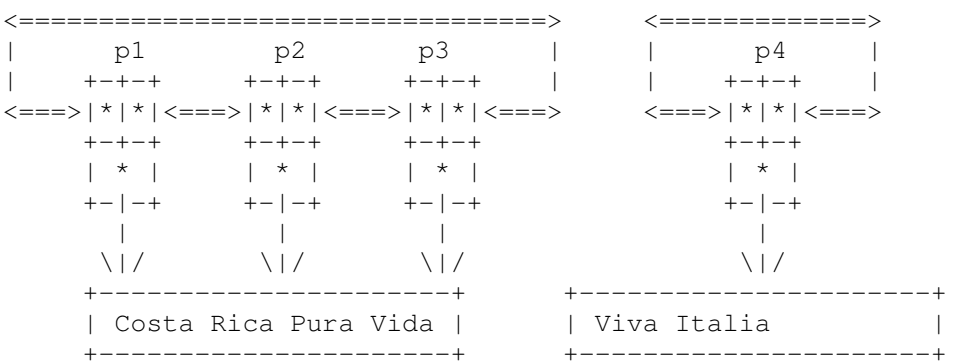


adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la
Computación e Informática

Doble enlace para compartir objetos
Afuera



adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

Es caro copiar el objeto retornado

```
/// Calcula y retorna \c (A * B).
Matrix operator* ( const Matrix& A, const Matrix& B ) {
    Matrix Resultado...
    // ... algoritmo de multiplicación de matrices
    return Resultado;
}
```

Es barato retornar un puntero

```
/// Calcula y retorna un puntero a \c (A * B).
Matrix* operator* ( const Matrix& A, const Matrix& B ) {
    Matrix * Resultado = new Matrix (...)
    // ...
    return Resultado;
}
```

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

Uso de lkptr<Matrix>

```
/// Calcula y retorna \c A+B
lkptr<Matrix> operator + (
    const lkptr<Matrix>& A,
    const lkptr<Matrix>& B
) {
    lkptr<Matrix> Res ( new Matrix(*A) );
    Res->Add(*B);
    return Res;
}
```

```
[protected]
/// Le suma a "*this" la matriz "O".
template<class E>
void Mx::Matrix<E>::add (  const Matrix<E> &  O  );
```

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

La clase "RefMatrix"

Implementación "RefMatrix.h"

En la implementación "RefMatrix.h" se han agregado únicamente las operaciones aritméticas que se quieren mejorar al hacerlas retornar referencias "lkptr<X>" en lugar de copias completas del objeto calculado.

- Los constructores y el destructor.
- Los operadores aritméticos " { + - * } ".
- La operación de "clone()" que retorna un puntero a una copia de la instancia.

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

```
/// Ejemplo de uso de referencia \c lkptr< RefMatrix<unsigned> >.
void use_lkptr( unsigned M, unsigned N ) {
    lkptr< RefMatrix<unsigned> > A ( new RefMatrix<unsigned>(M,N) );
    unsigned k = 0;
    for (unsigned i=0; i < A->rows(); ++i) {
        for (unsigned j=0; j < A->cols(); ++j) {
            A->at(i,j) = k++; // (*A)(i,j) = k++;
        }
    }

    lkptr< RefMatrix<unsigned> > B, C ( A ); // A y C comparten el valor
    assert( B == (void*)0 ); // B es un objeto nulo
    B = C; // A B y C comparten el valor
    B->reSize( B->cols(), B->rows() ); // todos fueron modificados
    assert( B == A );   // comparación de punteros
    assert( *B == *A ); // comparación de matrices
    B.reset( A->clone() ); // B tiene una copia de A
    B->reSize(N,M); // solo B cambia
    C = (C - C);
    assert( A->at(0,0) == 0 ); // porque C borró todo
    C = B + B - B;
    B->reSize( B->cols(), B->rows() ); // solo B cambia
    A = C * B; // ya ninguno comparte memoria
    assert( *A != *B && *B != *C && *C != *A ); // comparación de matrices
}
```

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

Diferencias Java vs C++

- En lugar de usar la notación de acceso al miembro "`A.m()`" es necesario usar "`A->m()`".
- Para obtener una copia del objeto referenciado hay que clonarlo, invocando tanto al operador "`new`" para obtener memoria dinámica como al constructor de copia de la clase base.
- Para obtener una copia del objeto referenciado no se puede usar el operador de asignación, por al hacerlo lo que se lograr en que 2 punteros "`lkptr<X>`" compartan el mismo objeto.

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la
Computación e Informática

```
/** Convierte a \c "T" en su sub-árbol espejo.
   - Recursivamente, sus hijos quedan en orden inverso del
     orden en el que aparecían originalmente en \c "T".
   \code   a                   a
          / \                 / \
         /   \               /   \
        b     e             e     b
       / \   / \           / \   / \
      f   h i   k         k   i h   f
               / \       / \
              l   m     m   l
                 / \   / \
                n   o o   n
    \endcode
*/
template <class E> void mirror( Bin_Tree<E> & T ) {
     if ( T.isEmpty() ) {
        return; // se sale si el árbol está vacío
    }
    // intercambia los hijos
    Bin_Tree<E> Left  = T.left();  // sostiene a cada hijo
    Bin_Tree<E> Right = T.right();
    T.makeLeftChild( Right ); // Pone el hijo derecho a la izquierda
    T.makeRightChild( Left ); // Pone el hijo izquierdo a la derecha
    mirror( Left  );
    mirror( Right ); // recursivamente modifica los hijos
}
```

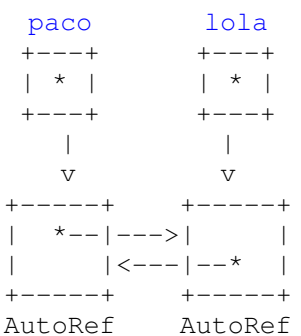
adolfo.dimare@ecci.ucr.ac.cr

http://www.di-mare.com/adolfo/p/lkptr.htm

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

Peligro de usar referencias cíclicas

Las referencias cíclicas o circulares pueden producir problemas si se usan punteros inteligentes, pues la destrucción del objeto referenciado puede producir un llamado recursivo infinito.



adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

Control de concurrencia

```
#include <MySemaphore.h>

template <class X> class lkptr {
   // ...
private: /// Asegura que sólo 1 proceso puede modificar.
   static Semaphore m_semaphore : Semaphore() { /* ... */ }
   // ...

        void fast_release() {
            m_Semaphore.wait(); // bloquea a los demás
        // ... libera el puntero inteligente ...
            m_Semaphore.signal(); // otros pueden modificar
        }
        void acquire(lkptr* r) throw() {
            m_Semaphore.wait(); // bloquea a los demás
            m_ptr = r->m_ptr;
            m_next = r->m_next;
            m_next->m_prev = this;
            m_prev = r;
            r->m_next = this;
            m_Semaphore.signal(); // otros pueden modificar
        }
}; // class lkptr<X>
```

adolfo.dimare@ecci.ucr.ac.cr

http://www.di-mare.com/adolfo/p/lkptr.htm

Conclusiones

- Para usar "`lkptr<X>`" basta incluir un único archivo de encabezado "`lkptr.h`" en contraposición a utilizar una biblioteca o ambiente complete que se encargue de la recolección de basura.
- El estilo de programación que "`lkptr<X>`" soporta permite utilizar el código C++ existente sin necesidad de hacerle cambios profundos.
- No se requiere aprender un nuevo paradigma de programación para poder usar objetos grandes con comodidad si se manipulan a través de punteros inteligentes "`lkptr<X>`".
- La existencia de un objeto compartido por referencias "`lkptr<X>`" no va más allá de la última referencia activa hacia el objeto, lo que garantiza una recuperación eficiente de la memoria dinámica que ya no está en uso.

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

Conclusiones

- Debido a que "`lkptr<X>`" no es un recolector de basura, no impone retardos asincrónicos con la aplicación y tampoco introduce un componente aleatorio en la duración de la ejecución.
- Los programas que usan "`lkptr<X>`" son deterministas porque bajo las mismas condiciones su tiempo de ejecución siempre es el mismo.
- Para la programación de sistemas incrustados o de tiempo real conviene usar "`lkptr<X>`" porque la implementación no usa memoria dinámica para almacenar cada referencia.
- La implementación de "`lkptr<X>`" no levanta excepciones porque no necesita adquirir memoria dinámica. El único caso en que una excepción puede ocurrir es si el destructor de la clase referenciada levanta una excepción, en cuyo caso el manejo usual de excepciones de C++ permite lidiar con el problema.
- Utilizando un semáforo es posibole usar "`lkptr<X>`" en ambientes de programación concurrente.

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm

lkptr.h

Universidad de Costa Rica
Escuela de Ciencias de la
Computación e Informática

Código fuente

- **http://www.di-mare.com/adolfo/p/lkptr.htm**
- **http://www.di-mare.com/adolfo/p/lkptr/index.html**
- **http://www.di-mare.com/adolfo/p/lkptr/lkptr.zip**

¡ Muchas gracias !

adolfo.dimare@ecci.ucr.ac.cr
http://www.di-mare.com/adolfo/p/lkptr.htm