Tipo de Dato Abstracto
Cadena

Prof. Andrés Arcia

Programación 2

Introduccion

- Una cadena o string es una hilera o cola de caracteres.
- El concepto sugiere un principio y un final, entonces deben ser demarcados.
- Imagina cual es el tipo de dato base para esta abstracción?
- Qué cree que se puede hacer con un conjunto de elementos del tipo cadena?
- Cómo haría Usted para atacar la escala? Hasta donde deberia llegar una cadena?

Introduccion

- Según Stroustroup, la experiencia ha demostrado que es imposible diseñar la clase Cadena perfecta, pues no se puede satisfascer a todos los clientes.
- Por lo pronto la clase String de la STL esta muy bien.
- Vamos a Definir un Tipo de Dato Abstracto y una implantación de String.

TDA Cadena

- Un tipo de dato abstracto para una cadena, debe contemplar:
  - Constructores
  - Extracción de subcadenas
  - Concatenación
  - Busqueda
  - Asignación (Reemplazo)
  - y hasta podría incluir:
  - Sustitución (busqueda y reemplazo)
  - Busqueda avanzada con expresiones regulares.

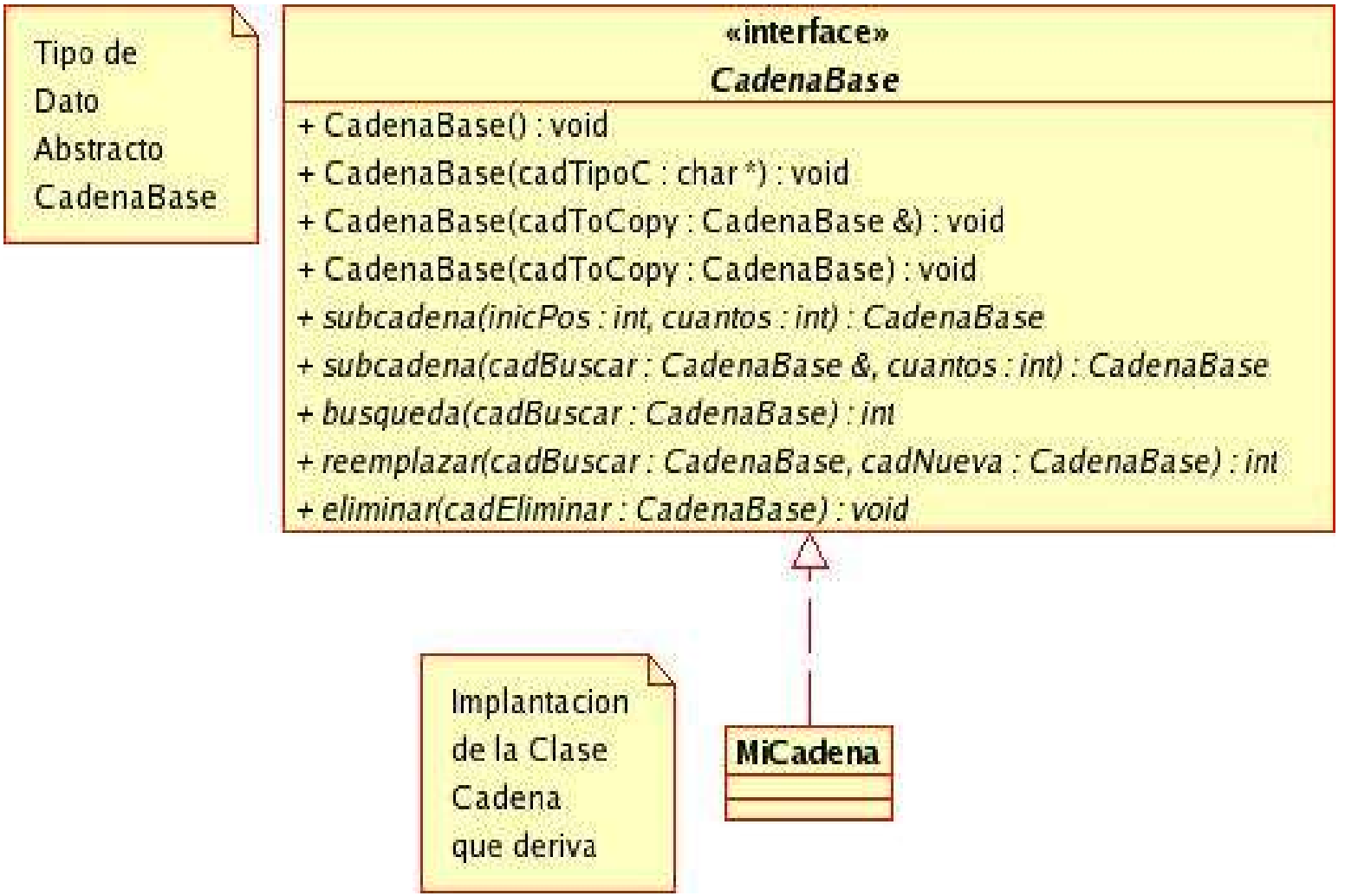
TDA Cadena

```
class CadenaBase
{   struct RepositorioCadena;
    public:
     // CONSTRUCTORES
    CadenaBase()
    CadenaBase(char *)
    CadenaBase(CadenaBase &)
    CadenaBase(CadenaBase)
```

TDA Cadena

```
// SUBCADENAS
      virtual CadenaBase subcadena(int inicPos, int cuantos)
      virtual CadenaBase subcadena(CadenaBase &, int cuantos)
// BUSQUEDA
     virtual int Busqueda(CadenaBase &)
// REEMPLAZO
      virtual void reemplazar(CadenaBase &, CadenaBase &)
// ELIMINACION
      virtual void eliminar(int inicPos, int cuantos)
// DESTRUCTOR
      ~CadenaBase() };
```

Diagrama OMT



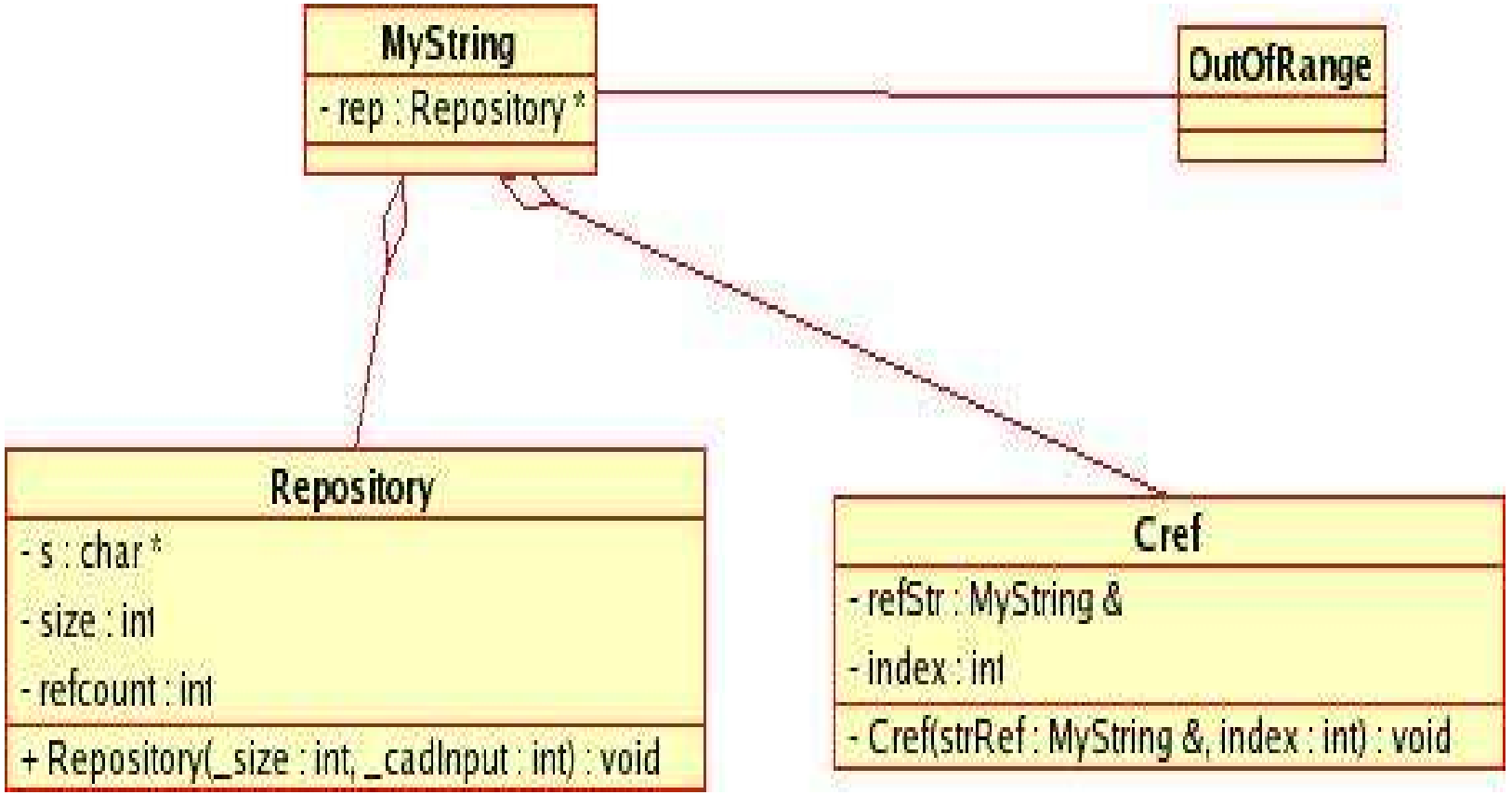
TDA Cadena

- En la implantación que veremos hay tres componentes básicos:
  - Una estructura que representa la cadena en si.
  - Una clase que ayudará a diferenciar las escrituras de las lecturas: Cref
  - Excepciones. La más común será la de fuera de rango.

TDA Cadena

```
class MyString {
        struct Srep;
        Srep * rep;
public:
         class Cref;
         class OutOfRange { }; // clase vacia, para la
   excepción.
};
```

Diagrama OMT



TDA Cadena

```
struct MyString::Srep {
        char * s;   int size;  int n;

Srep (int nsize, const char * p)
{ n=1;
   size=nsize;
   s = new char[size+1];
   strcpy(s, p);
}
~Srep() { delete[] s; }
```

TDA Cadena

```
Srep * get_own_copy()
{ if (n==1) return this;
   n--;
   return new Srep(size, s);
  }
void assign(int nsize, const char* p)
{ if (size != nsize)
     { delete[] s;
       size = nsize;
       s = new char [size+1]; }
      strcpy(s, p); }
```

TDA Cadena

```
private:
    Srep(const Srep&) // previene la copia.
    Srep& operator= (const Srep&);
};
public:
MyString::MyString() { rep = new Srep(0, “ “) }
MyString::MyString(MyString &x) // Copy constructor
{ x.rep->n++
  rep = x.rep };
```

TDA Cadena

MyString::MyString() // La cadena vacia es el valor por omisión.

{ rep = new Srep(0, “”); }

MyString::MyString(MyString &x) // Constructor copia

{ x.rep->n++;

  rep=x.rep; }

MyString::~MyString()

{ if (--rep->n == 0) delete rep; // llama al destructor de rep

}

MyString & **MyString::operator=**(const String &x) // Asignación copia

{ x.rep->n++;

  if (--rep->n == 0) delete rep;

rep = x.rep;   // compartir la representación

return *this; }

TDA Cadena

Operaciones de Pseudo Copia:

MyString::MyString(const char *s)

{ rep = new Srep(strlen(s),s); }

MyString & **MyString::operator=**(const char* s)

```
{ if (rep->n == 1)
      rep->assign(strlen(s),s);
  else { rep->n—; // copias anteriores se disminuyen en 1 unidad
         rep = new Srep(strlen(s),s); // reemplazo }
  return *this;
}
```

TDA Cadena

```
class MyString {

void check(int i) const { if (i<0 || rep->size<=i) throw
    OutOfRange(); }

char read(int i) const {return rep->s[i]; }

void write(int i, char c) {rep = rep->get_own_copy(); rep->s[i]
    =c; }

Cref operator[](int i) {check(i); return Cref(*this, i); }

char operator[](int i) const {check(i); return rep->s[i];}


int size() const {return rep->size; }

};
```

TDA Cadena

```
class MyString::Cref {
friend class MyString;
        MyString & s;
        int i;
        Cref(MyString &ss, int ii) : s(ss), i(ii) { }
public:
        operator char() { return s.read(i); } // lectura de valor
        void operator= (char c) {s.write(i,c); }// cambio de valor
       };
```

Ejemplo

Para el siguiente ejemplo, determine que funciones se estan llamando.

```
void f(MyString left, const MyString & constright)
{
  int c1 = left[1]; // operator[](1).operator char()
  left[1] = 'c'; // operator[](1).operator=('c')
  int c2 = constright[1]; // operator[](1)
  constright[1] = 'd'; // error, asignacion a objeto constante
}
```

Otras funciones

```
class MyString {
MyString & operator += (const MyString &)
MyString & operator += (const char *)
friend bool operator == (const MyString & x, const MyString & y)
{ return strcmp(x.rep->s, y.rep->s) == 0; }
};
```

Diferentes Implantaciones

- Una cadena puede implantarse:
  - Con un par de variables: un repositorio (arreglo) y una variable entera para indicar el tamaño de la cadena.
  - Con delimitador: un caracter especial es designado como fin de cadena.
  - Cadenas dinámicas: pueden crecer tanto como se quiera, teniendo en cuenta el mecanismo para juntarlas.

Ejercicios

- Haga las siguientes funciones:
  - Invierta una cadena
  - Separe una cadena en dos, una con las posiciones pares y la otra con las impares.
  - Rote una cadena dado un numero de espacios.
  - Convierta una frase en un titulo. Mayuscula la primera letra y minusculas el resto.
  - Implante una cadena a partir del TDA cadena.