Anotaciones sobre operadores en TDAs

(en rojo marco un par de cosillas que comenté en clase sobre postincrementoy estaban mal, el resto es un recordatorio y cosas nuevas)

–

Alguien me comentó que faltaría la semana del 10 al 16, como es importante que todos comprendáis esta parte he preparado esta presentación.

–

Podéis obtener el código fuente en la web

Anotaciones sobre operadores en TDAs

Trabajamos sobre nuestro TDA Entero.

Como **simplificación** para facilitar la comprensión, en lugar de poder trabajar con enteros de hasta cien dígitos y por tanto necesitar vectores para representarlos, la **representación** va a hacerse en un int, por lo tanto tendríamos las mismas restricciones de longitud.

Diferenciaré entre los métodos dentro de la clase y fuera de la clase en la implementación de los mismos (en el código están implementados directamente dentro):

```
Entero Entero::operator++(int val) { //dentro

Entero operator++(Entero &e,int val) { //fuera
```

Anotaciones sobre operadores en TDAs

Postincremento: e++ (devuelve valor actual e incrementa)

```
Entero Entero::operator++(int val) { // e++ (e++0) //DENTRO
  // OJO, no puede devolver referencia, b dejaría de existir al
  //   salir.
  Entero b;
  b = *this;
  a+=1; //OJO, no val, val vale siempre 0.
  return b;
}

Entero &operator++(Entero &e,int val) {  // e++ (e++0) //FUERA
  // OJO, no puede devolver referencia, b dejaría de existir al
  //   salir.
  Entero b;
  b = e;
  e = e+1; //OJO, no podemos acceder a a, estamos fuera.
  return b;
}

//Este operador se puede declarar tanto DENTRO como FUERA
```

Anotaciones sobre operadores en TDAs

Preincremento: ++e (incrementa y devuelve valor final)

```
Entero &Entero::operator++() {  // ++e //DENTRO
  a++;
  return *this;
}

Entero &operator++(Entero &e) {  // ++e //FUERA
  e = e+1; //OJO, no podemos acceder a a, estamos fuera.
  return e;
}
```

//Este operador se puede declarar tanto DENTRO como FUERA

Anotaciones sobre operadores en TDAs

Operador de asignación / Constructor por copia, por defecto:

Si no los definimos, copian correctamente todos los miembros declarados (variables y vectores de longitud definida (como int bla[20];)), usando el operador =.

Copian **tal cual** los valores de los punteros, por lo que acabarán **apuntando al mismo objeto o lugar**.

Si no queremos que esto ocurra, debemos definir nuestros propios métodos que realicen las reservas de memoria dinámica oportunas, o anular los por defecto declarándolos privados:

```
private:
Clase Clase(const Clase& a);
Clase operator=(const Clase& a);
```

Anotaciones sobre operadores en TDAs

Operador de asignación:

```
Entero &Entero::operator=(const Entero & e) { //CUIDADO, ¡ERROR!
  this->a = e.a;
  return e;
}
```

En este caso estamos devolviendo un const Entero & como Entero &.
¡Eso no podemos hacerlo, perdería la condición de constante!

Anotaciones sobre operadores en TDAs

Soluciones, **en principio** válidas las tres:

```
const Entero &Entero::operator=(const Entero & e) {
  this->a = e.a; //Devolvemos como constante también.
  return e;
}

Entero &Entero::operator=(const Entero & e) {
  this->a = e.a; //O devolvemos el actual.
  return *this;
}

Entero Entero::operator=(const Entero & e) {
  this->a = e.a; //O devolvemos el actual.
  return *this;
}
```

Anotaciones sobre operadores en TDAs

Revisamos la primera opción:

```
const Entero &Entero::operator=(const Entero & e) {
  this->a = e.a; //Devolvemos como constante también...
  return e;
}

Si tenemos: Entero a,b,c;
No permite: c=((b=a)++)
Si queremos que no lo permita, esta opción es válida.
```

Anotaciones sobre operadores en TDAs

Revisamos la segunda opción:

```
Entero &Entero::operator=(const Entero & e) {
  this->a = e.a; //O devolvemos referencia al actual...
  return *this;
}

Si tenemos: Entero a,b,c;
Permite: c=(b=a)++ pero, OJO, se modificaría b!!
¡Esta opción no es válida!
```

Anotaciones sobre operadores en TDAs

Revisamos la tercera opción:

```
Entero Entero::operator=(const Entero & e) {
  this->a = e.a; //O devolvemos copia modificable del actual.
  return *this;
}
```

Si tenemos: Entero a,b,c;
Permite: c=(b=a)++ y b no se modifica.
Esta solución es correcta si queremos que se pueda incrementar el valor temporal devuelto.

Ojo, si la copia de este TDA es muy costosa puede no interesarnos permitirlo, porque se realizaría una en cada sentencia de asignación.

Anotaciones sobre operadores en TDAs

Operadores de conversión:

```
Entero::operator int() { //Ojo, no se pone tipo al principio.
  return a;
}

Esto permite conversiones explícitas: cout << (int)e << endl;
Y conversiones implícitas (con cuidado): cout << e << endl;

Entero::operator int*() { //Ojo, no se pone tipo al principio.
  return &a;
}

Permite conversiones explícitas: cout << (*(int*)e) << endl;
Y también implícitas (con cuidado).
```

Anotaciones sobre operadores en TDAs

Operadores de conversión:

Cuidado, si ponemos const para el objeto actual

```
Entero::operator int*() const {
  return &a;
}
```

No compilaría, porque a través de ese puntero (a un entero no constante) podría modificarse el contenido del TDA.

Sí valdrían (indistintamente, ya vimos que estos const afectan al int:

```
Entero::operator const int*() const {
  return &a;
}

Entero::operator int const*() const {
  return &a;
}
```

Anotaciones sobre operadores en TDAs

Operadores de entrada salida:

Ejemplo de salida, muestra: “Entero(valor)”.
¡Debe ir declarado **FUERA** de la clase!

```
ostream &operator<<(ostream &out,const Entero &e) {
  out << "Entero(" << (int)e << ")";
  return out;
}
```

Ejemplo de uso:

```
int main() {
  Entero e;
  e++;
  ++e;
  cout << e << endl;
}
```

Muestra: Entero(2)

Anotaciones sobre operadores en TDAs

Operadores *elemento y &elemento:

```
Entero Entero::operator*();
Entero *Entero::operator&();
```