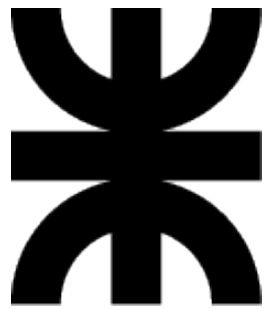
Informática II
Objetos, funciones y datos miembros `const`

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Objetos y funciones miembros `const`

Si se necesita un objeto de la clase `Hora` llamado `mediodia` cuyo valor no pueda ser modificado

```
const Hora mediodia(12, 0, 0);
```

Objetos y funciones miembros `const`

Si se necesita un objeto de la clase `Hora` llamado `mediodia` cuyo valor no pueda ser modificado

```
const Hora mediodia(12, 0, 0);
```

- Un objeto `const` no se puede modificar por asignación así que es necesario inicializarlo

Objetos y funciones miembros `const`

Si se necesita un objeto de la clase `Hora` llamado `mediodia` cuyo valor no pueda ser modificado

```
const Hora mediodia(12, 0, 0);
```

- Un objeto `const` no se puede modificar por asignación así que es necesario inicializarlo
- No se pueden llamar a funciones miembros de objetos `const` a menos que dicha función se declare también `const`
- Las funciones miembros `const` no pueden modificar el objeto

Objetos y funciones miembros `const`

Si se necesita un objeto de la clase `Hora` llamado `mediodia` cuyo valor no pueda ser modificado

```
const Hora mediodia(12, 0, 0);
```

- Un objeto `const` no se puede modificar por asignación así que es necesario inicializarlo
- No se pueden llamar a funciones miembros de objetos `const` a menos que dicha función se declare también `const`
- Las funciones miembros `const` no pueden modificar el objeto

Una función se debe especificar `const` en su prototipo y en su definición

```
int NombreClase::obtieneValor() const
{
  return datoMiembroPrivado;
}
```

Objetos y funciones miembros `const`

Si se necesita un objeto de la clase `Hora` llamado `mediodia` cuyo valor no pueda ser modificado

```
const Hora mediodia(12, 0, 0);
```

- Un objeto `const` no se puede modificar por asignación así que es necesario inicializarlo
- No se pueden llamar a funciones miembros de objetos `const` a menos que dicha función se declare también `const`
- Las funciones miembros `const` no pueden modificar el objeto

Una función se debe especificar `const` en su prototipo y en su definición

```
int NombreClase::obtieneValor() const
{
  return datoMiembroPrivado;
}
```

Es recomendable declarar como `const` a todas las funciones miembros que no necesitan modificar el objeto

Dato miembro `const` – inicialización

```
1  class Incremento {
2    public:
3      Incremento(int c = 0, int i = 1);
4      void sumaIncremento() { cuenta += incremento; }
5      void imprime() const;
6
7
8
9
10 };
```

Dato miembro `const` – inicialización

```
1  class Incremento {
2    public:
3      Incremento(int c = 0, int i = 1);
4      void sumaIncremento() { cuenta += incremento; }
5      void imprime() const;
6
7    private:
8      int cuenta;
9      const int incremento; // dato miembro const
10 };
```

Dato miembro `const` – inicialización

```
1  class Incremento {
2    public:
3      Incremento(int c = 0, int i = 1);
4      void sumaIncremento() { cuenta += incremento; }
5      void imprime() const;
6
7    private:
8      int cuenta;
9      const int incremento; // dato miembro const
10 };
```

```
1  // Constructor para la clase Incremento
2  Incremento::Incremento(int c, int i)
3  {
4    cuenta = c;
5    incremento = i;
6  }
```

Dato miembro `const` – inicialización

Error de compilación al intentar modificar un dato miembro constante

```
incremento.cpp: In constructor 'Incremento::Incremento(int, int)':
incremento.cpp:21:1: error: uninitialized const member in
   'const int' [-fpermissive]
 Incremento::Incremento(int c, int i)
 ^
incremento.cpp:17:15: note: 'const int Incremento::incremento'
  should be initialized
                const int incremento;   // dato miembro const
                          ^
incremento.cpp:24:14: error: assignment of read-only member
  'Incremento::incremento'
         incremento = i;
                    ^
```

Dato miembro `const` – inicialización

Error de compilación al intentar modificar un dato miembro constante

```
incremento.cpp: In constructor 'Incremento::Incremento(int, int)':
incremento.cpp:21:1: error: uninitialized const member in
   'const int' [-fpermissive]
 Incremento::Incremento(int c, int i)
 ^
incremento.cpp:17:15: note: 'const int Incremento::incremento'
  should be initialized
               const int incremento;   // dato miembro const
                         ^
incremento.cpp:24:14: error: assignment of read-only member
  'Incremento::incremento'
         incremento = i;
                    ^
```

¿Cómo inicializar un dato miembro `const` si no es posible realizar una asignación en el constructor?

Dato miembro `const` – inicialización

Se tiene que modificar el constructor de `Incremento` de la sig. manera:

```
Incremento::Incremento(int c, int i) : incremento(i)
{
  cuenta = c;
}
```

Dato miembro `const` – inicialización

Se tiene que modificar el constructor de `Incremento` de la sig. manera:

```
Incremento::Incremento(int c, int i) : incremento(i)
{
  cuenta = c;
}
```

- La notación `:incremento(i)` inicializa `incremento` al valor de `i`

Dato miembro `const` – inicialización

Se tiene que modificar el constructor de `Incremento` de la sig. manera:

```
Incremento::Incremento(int c, int i) : incremento(i)
{
  cuenta = c;
}
```

- La notación `:incremento(i)` inicializa `incremento` al valor de `i`
- Si se necesitan varios inicializadores se tiene que utilizar una lista separada por comas después de los dos puntos

Dato miembro `const` – inicialización

Se tiene que modificar el constructor de `Incremento` de la sig. manera:

```
Incremento::Incremento(int c, int i) : incremento(i)
{
  cuenta = c;
}
```

- La notación `:incremento(i)` inicializa `incremento` al valor de `i`
- Si se necesitan varios inicializadores se tiene que utilizar una lista separada por comas después de los dos puntos
- Todos los datos miembros *pueden* inicializarse utilizando la sintaxis anterior, pero los datos miembros `const` *deben* inicializarse de esta manera

Dato miembro `const` – inicialización

La sintaxis `incremento(i)` puede verse como crear un objeto `incremento` (aún cuando sea un tipo de dato predefinido), pasándole al constructor el valor de inicialización, en este caso `i`.

Dato miembro `const` – inicialización

La sintaxis `incremento(i)` puede verse como crear un objeto `incremento` (aún cuando sea un tipo de dato predefinido), pasándole al constructor el valor de inicialización, en este caso `i`.

Ver código fuente ejemplo de D&D 4° ed. `fig17_02.cpp` y `fig17_03.cpp`.