Constructores y Lista de Inicialización (repaso)

Algoritmos y Estructuras de Datos II

Constructor: ejemplo

```
class A {
  public:
    ...
  private:
    int x_;
    string s_;
    vector<float> fs_;
};

int main() {
    int j = 10;
    A a = A();
}
```

a.x_	??
a.s_	??
a.fs_	??
j	10
	pila

Constructor: concepto

- Los constructores son funciones especiales para inicializar una nueva instancia de un tipo.
- Se escriben con el nombre del tipo.
- No tienen tipo de retorno (está implícito).
- Tres formas de constructores:
  - Por defecto: `Horario();`
  - Por copia: `Horario(Horario otro_horario);`[1]
  - Con parámetros (el resto):

```
Horario(int hora_, int min_, int seg_);
Horario(int hora_);
```

- Cuando se define un constructor desaparece el constructor sin parámetros implícito (se lo puede definir explícitamente).

[1] La aridad es levemente distinta, ver después...

Constructor: declaraciones

- Constructor por defecto: `T::T()` No recibe parametros
- Constructor por copia: `T::T(const T&)`
- Destructor: `T::~T()`
- Operador asignación: `T::operator=(const T&)`

Diferencia solo se llama cuando se inicializa una variable mientras que el operador asignación hay que asumir que la variable esta inicializada

Constructor por copia

¿Qué pasaría si Lista no tuviera constructor por copia?

```
class Lista {
  public:
    Lista();
    Lista(const Lista& otra);

    void agregarAtras(T&);
    int longitud() const;
    ...

  private:
    struct Nodo {
        T valor;
        Nodo* siguiente;
    }

    Nodo* primero_;
}
```

```
int main() {
    Lista<int> l1;
    l1.agregarAtras(1);
    Lista<int> l2(l1);
    l2.agregarAtras(2);
    l1.longitud(); // ??
}
```

Constructor por copia

¿Qué pasaría si Lista no tuviera constructor por copia?

```
class Lista {
  public:
    Lista();
    Lista(const Lista& otra);

    void agregarAtras(T&);
    int longitud() const;
    ...

  private:
    struct Nodo {
        T valor;
        Nodo* siguiente;
    }

    Nodo* primero_;
}
```

```
int main() {
    Lista<int> l1;
    l1.agregarAtras(1);
    Lista<int> l2(l1);
    l2.agregarAtras(2);
    l1.longitud(); // ??
}
```

¡Pizarrón!

¿Dónde, dónde está el constructor por copia?

¿Dónde se llama al constructor por copia en este ejemplo?

```
int maximo(Lista<int> l) {
    int max = *(l.first());
    for (int x : l) {
        if (x > max) {
            max = x;
        }
    }
    return max;
}

Lista<int> l1;
l1.agregarAtras(1);
l1.agregarAtras(3);
l1.agregarAtras(2);
int m = maximo(l1);
```

¿Y en este?

```
Lista<int> l1;
l1.agregarAtras(1);
Lista<int> l2 = l1;
```

¿Y en esteeee?

```
Lista<int> rango(int desde, int hasta) {
    Lista<int> ret;
    for (int i = desde; i < hasta; i++) {
        ret.agregarAtras(i);
    }
    return ret;
}

Lista r = rango(5, 25);
```

¿Y en este otro?

```
Lista<int> l1;
l1.agregarAtras(1);
l1.agregarAtras(10);
Lista<int> l2;
l2 = l1;
```

¿Qué pasaría si Lista no tuviera operador de asignación?

```
class Lista {
  public:
    Lista();
    Lista(const Lista& otra);
    Lista& operator=(const Lista& otra);

    void agregarAtras(T&);
    int longitud() const;
    ...

  private:
    struct Nodo {
        T valor;
        Nodo* siguiente;
    }

    Nodo* primero_;
}
```

```
int main() {
    Lista<int> l1;
    l1.agregarAtras(1);
    Lista<int> l2;
    l2.agregarAtras(10);
    l2 = l1;
    l2.agregarAtras(2);
    l1.longitud(); // ??
}
```

¿Qué pasaría si Lista no tuviera operador de asignación?

```
class Lista {
  public:
    Lista();
    Lista(const Lista& otra);
    Lista& operator=(const Lista& otra);

    void agregarAtras(T&);
    int longitud() const;
    ...

  private:
    struct Nodo {
        T valor;
        Nodo* siguiente;
    }

    Nodo* primero_;
}
```

```
int main() {
    Lista<int> l1;
    l1.agregarAtras(1);
    Lista<int> l2;
    l2.agregarAtras(10);
    l2 = l1;
    l2.agregarAtras(2);
    l1.longitud(); // ??
}
```

¡Pizarrón!

Constructor por copia

¿Necesito constructor por copia para este caso?

```
class MaximoRapido {
  public:
    MaximoRapido(const Lista<int>& l);
    int maximo() const;

  private:
    Lista<int> lista_;
    Lista<int>::iterator max_;
}

MaximoRapido::MaximoRapido(const Lista<int>& l) {
    lista_ = l;
    // buscar el máximo
}
```

Constructor por copia

¿Necesito constructor por copia para este caso?

```
class MaximoRapido {
  public:
    MaximoRapido(const Lista<int>& l);
    int maximo() const;

  private:
    Lista<int> lista_;
    Lista<int>::iterator max_;
}

MaximoRapido::MaximoRapido(const Lista<int>& l) {
    lista_ = l;
    // buscar el máximo
}
```

¡Pizarrón!

Lista de inicialización

```
class Fecha {
 public:
  // pre: anio > 0, mes \in [1, 12],
  ↪  dia \in [1, diasEnMes(anio,
  ↪  mes)]
  Fecha(Anio anio, Mes mes, Dia dia);
  Fecha(Fecha fecha, Periodo periodo);
  Fecha(const Fecha& o);

  Anio anio() const;
  Mes mes() const;
  Dia dia() const;

  bool operator==(Fecha o) const;
  bool operator<(Fecha o) const;

  void sumar_periodo(Periodo p);

 private:
  Anio anio_;
  Mes mes_;
  Dia dia_;

  void ajustar_fecha();
  void sumar_anios(Anio anios);
  void sumar_meses(Mes meses);
  void sumar_dias(Dia dias);
};
```

```
class Intervalo {
 public:
  // pre: desde < hasta
  Intervalo(Fecha desde, Fecha hasta);
  Intervalo(Fecha desde, Periodo periodo);

  Fecha desde() const;
  Fecha hasta() const;

  int enDias() const;

 private:
  Fecha desde_;
  Fecha hasta_;
};
```

```
class Periodo {
 public:
  Periodo(int anios, int
  ↪  meses, int dias);

  int anios() const;
  int meses() const;
  int dias() const;

 private:
  int anios_;
  int meses_;
  int dias_;
};
```

```
Intervalo::Intervalo(Fecha desde, Fecha hasta) {
    desde_ = desde;
    hasta_ = hasta;
}
```

Lista de inicialización

```
class Fecha {
 public:
  // pre: anio > 0, mes \in [1, 12],
  ↪  dia \in [1, diasEnMes(anio,
  ↪  mes)]
  Fecha(Anio anio, Mes mes, Dia dia);
  Fecha(Fecha fecha, Periodo periodo);
  Fecha(const Fecha& o);

  Anio anio() const;
  Mes mes() const;
  Dia dia() const;

  bool operator==(Fecha o) const;
  bool operator<(Fecha o) const;

  void sumar_periodo(Periodo p);

 private:
  Anio anio_;
  Mes mes_;
  Dia dia_;

  void ajustar_fecha();
  void sumar_anios(Anio anios);
  void sumar_meses(Mes meses);
  void sumar_dias(Dia dias);
};
```

```
class Intervalo {
 public:
  // pre: desde < hasta
  Intervalo(Fecha desde, Fecha hasta);
  Intervalo(Fecha desde, Periodo periodo);

  Fecha desde() const;
  Fecha hasta() const;

  int enDias() const;

 private:
  Fecha desde_;
  Fecha hasta_;
};
```

```
class Periodo {
 public:
  Periodo(int anios, int
  ↪  meses, int dias);

  int anios() const;
  int meses() const;
  int dias() const;

 private:
  int anios_;
  int meses_;
  int dias_;
};
```

```
Intervalo::Intervalo(Fecha desde, Fecha hasta) {
    desde_ = desde;
    hasta_ = hasta;
}
```

```
const_Intervalo.cpp: In constructor 'Intervalo::Intervalo(Fecha, Fecha)':
const_Intervalo.cpp:59:46: error: no matching function for call to 'Fecha::Fecha()'
 Intervalo::Intervalo(Fecha desde, Fecha hasta) {
                                              ^
```

Lista de inicialización

```
class Fecha {
 public:
  // pre: anio > 0, mes \in [1, 12],
 ↪  dia \in [1, diasEnMes(anio,
 ↪  mes)]
  Fecha(Anio anio, Mes mes, Dia dia);
  Fecha(Fecha fecha, Periodo periodo);
  Fecha(const Fecha& o);

  Anio anio() const;
  Mes mes() const;
  Dia dia() const;

  bool operator==(Fecha o) const;
  bool operator<(Fecha o) const;

  void sumar_periodo(Periodo p);

 private:
  Anio anio_;
  Mes mes_;
  Dia dia_;

  void ajustar_fecha();
  void sumar_anios(Anio anios);
  void sumar_meses(Mes meses);
  void sumar_dias(Dia dias);
};
```

```
class Intervalo {
 public:
  // pre: desde < hasta
  Intervalo(Fecha desde, Fecha hasta);
  Intervalo(Fecha desde, Periodo periodo);

  Fecha desde() const;
  Fecha hasta() const;

  int enDias() const;

 private:
  Fecha desde_;
  Fecha hasta_;
};
```

```
class Periodo {
 public:
  Periodo(int anios, int
 ↪  meses, int dias);

  int anios() const;
  int meses() const;
  int dias() const;

 private:
  int anios_;
  int meses_;
  int dias_;
};
```

```
Intervalo::Intervalo(Fecha desde, Fecha hasta)
  : desde_(desde), hasta_(hasta) {};
```

Lista de inicialización

```
class Fecha {
 public:
  // pre: anio > 0, mes \in [1, 12],
  ↪  dia \in [1, diasEnMes(anio,
  ↪  mes)]
  Fecha(Anio anio, Mes mes, Dia dia);
  Fecha(Fecha fecha, Periodo periodo);
  Fecha(const Fecha& o);

  Anio anio() const;
  Mes mes() const;
  Dia dia() const;

  bool operator==(Fecha o) const;
  bool operator<(Fecha o) const;

  void sumar_periodo(Periodo p);

 private:
  Anio anio_;
  Mes mes_;
  Dia dia_;

  void ajustar_fecha();
  void sumar_anios(Anio anios);
  void sumar_meses(Mes meses);
  void sumar_dias(Dia dias);
};
```

```
class Intervalo {
 public:
  // pre: desde < hasta
  Intervalo(Fecha desde, Fecha hasta);
  Intervalo(Fecha desde, Periodo periodo);

  Fecha desde() const;
  Fecha hasta() const;

  int enDias() const;

 private:
  Fecha desde_;
  Fecha hasta_;
};
```

```
class Periodo {
 public:
  Periodo(int anios, int
  ↪  meses, int dias);

  int anios() const;
  int meses() const;
  int dias() const;

 private:
  int anios_;
  int meses_;
  int dias_;
};
```

```
Intervalo::Intervalo(Fecha desde, Periodo periodo)
  : desde_(desde), hasta_(desde) {

    hasta_.sumar_periodo(periodo);
};
```

Lista de inicialización

```
Fecha::Fecha(const Fecha& o) : anio_(o.anio()), mes_(o.mes()),
↪  dia_(o.dia()) {
}

Fecha::Fecha(const Fecha& o, Periodo p)
  : anio_(o.anio()), mes_(o.mes()), dia_(o.dia()) {
    sumar_periodo(p);
}

Intervalo::Intervalo(Fecha desde, Periodo periodo)
  : desde_(desde), hasta_(desde, periodo) {
};
```