Constness

Algoritmos y Estructuras de Datos II

Const en Variables

```
int main() {
    int x = 5;
    x++;
    cout << x << endl; // 6
    const int y = 5;
    y++; // error: increment of read-only variable 'y'
}
```

Variables

```
int main() {
    int x = 5;
    const int& y = x;
    x++;
    cout << y << endl; // 6
    y++; // error: increment of read-only reference 'y'
}
```

Variables

```
int main() {
    const int x = 5;
    int& y = x; // error: binding reference of type
    ↪  'int&' to 'const int' discards qualifiers
}
```

T foo(const P) (Const en parámetros)

```
int max(list<int> l) {
  int max = l.front();
  while (l.size() > 0) {
    if (l.front() > max) {
      max = l.front();
    }
    l.pop_front();
  }
  return max;
}

int main() {
  list<int> l;
  l.push_back(5);
  l.push_back(10);
  l.push_back(6);
  const list<int>& l2 = l;

  cout << "max: " << max(l) << endl; // 10
  cout << l.size() << endl; // 3
}
```

Const en parámetros

```
int max(list<int>& l) {
  int max = l.front();
  while (l.size() > 0) {
    if (l.front() > max) {
      max = l.front();
    }
    l.pop_front();
  }
  return max;
}

int main() {
  list<int> l;
  l.push_back(5);
  l.push_back(10);
  l.push_back(6);
  const list<int>& l2 = l;

  cout << "max: " << max(l) << endl; // 10
  cout << l.size() << endl; // 0
}
```

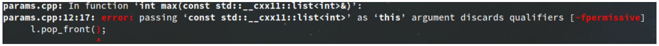
Const en parámetros

```
int max(list<int>& l) {
  int max = l.front();
  while (l.size() > 0) {
    if (l.front() > max) {
      max = l.front();
    }
    l.pop_front();
  }
  return max;
}

int main() {
  list<int> l;
  l.push_back(5);
  l.push_back(10);
  l.push_back(6);
  const list<int>& l2 = l;

  cout << "max: " << max(l2) << endl; //  error: binding
  ↪  reference of type ‘std::__cxx11::list<int>&’ to ‘const
  ↪  std::__cxx11::list<int>’ discards qualifiers
  cout << l.size() << endl; // 0
```

Const en parámetros

```cpp
int max(const list<int>& l) {
  int max = l.front();
  while (l.size() > 0) {
    if (l.front() > max) {
      max = l.front();
    }
    l.pop_front();
  }
  return max;
}
```

```cpp
int main() {
  list<int> l;
  l.push_back(5);
  l.push_back(10);
  l.push_back(6);
  const list<int>& l2 = l;

  cout << "max: " << max(l2) <<
  ↪  endl;
  cout << l.size() << endl; //
  ↪  0
}
```



```
params.cpp: In function ‘int max(const std::__cxx11::list<int>&)’:
params.cpp:12:17: error: passing ‘const std::__cxx11::list<int>’ as ‘this’ argument discards qualifiers [-fpermissive]
     l.pop_front();
                 ^
```

Const en parámetros

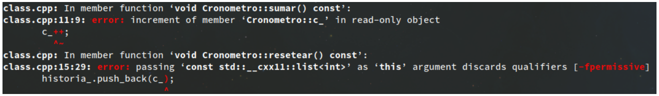
```
int max(const list<int>& l_) {
  list<int> l = l_;
  int max = l.front();
  while (l.size() > 0) {
    if (l.front() > max) {
      max = l.front();
    }
    l.pop_front();
  }
  return max;
}

int main() {
  list<int> l;
  l.push_back(5);
  l.push_back(10);
  l.push_back(6);
  const list<int>& l2 = l;

  cout << "max: " << max(l2) << endl; // 10
  cout << l.size() << endl; // 3
}
```

```
int max(const list<int>& l) {
  int m = 0;
  for (int x : l) {
    if (x > m) {
      m = x;
    }
  }
  return m;
}
```

const T foo(P) const (Const en clases)

```cpp
class Cronometro {
  public:
    Cronometro() : c_(0), historia_() {}

    void sumar() const {
      c_++;
    }

    void resetear() const {
      historia_.push_back(c_);
      c_ = 0;
    }

    list<int> historia() {
      return historia_;
    }

    int contador() {
      return c_;
    }

  private:
    int c_;
    list<int> historia_;
};
```

```cpp
int main() {
  Cronometro c;
  c.sumar();
  c.resetear();
}
```



```
class.cpp: In member function ‘void Cronometro::sumar() const’:
class.cpp:11:9: error: increment of member ‘Cronometro::c_’ in read-only object
       c_++;
         ^~
class.cpp: In member function ‘void Cronometro::resetear() const’:
class.cpp:15:29: error: passing ‘const std::__cxx11::list<int>’ as ‘this’ argument discards qualifiers [-fpermissive]
       historia_.push_back(c_);
                             ^
```

Classes

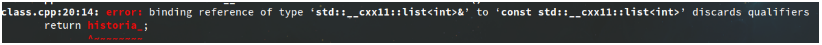
```cpp
class Cronometro {
  public:
    Cronometro() : c_(0), historia_() {}

    void sumar() {
      c_++;
    }

    void resetear() {
      historia_.push_back(c_);
      c_ = 0;
    }

    list<int> historia() {
      return historia_;
    }

    int contador() {
      return c_;
    }

  private:
    int c_;
    list<int> historia_;
};
```

```cpp
int main() {
  Cronometro c;
  c.sumar();
  const Cronometro& cr = c;
  cr.historia();
  // error: passing ‘const
  ↪   Cronometro’ as ‘this’
  ↪   argument discards
  ↪   qualifiers [-fpermissive]
}
```

Classes

```
class Cronometro {
  public:
    Cronometro() : c_(0), historia_() {}

    void sumar() {
      c_++;
    }

    void resetear() {
      historia_.push_back(c_);
      c_ = 0;
    }

    list<int>& historia() const {
      return historia_;
    }

    int contador() {
      return c_;
    }

  private:
    int c_;
    list<int> historia_;
};
```

```
int main() {
  Cronometro c;
  c.sumar();
  const Cronometro& cr = c;
  list<int>& l =
  ↪   cr.historia();
}
```



```
class.cpp:20:14: error: binding reference of type ‘std::__cxx11::list<int>&’ to ‘const std::__cxx11::list<int>’ discards qualifiers
      return historia_;
             ^~~~~~~~~
```

Classes

```
class Cronometro {
  public:
    Cronometro() : c_(0), historia_() {}

    void sumar() {
      c_++;
    }

    void resetear() {
      historia_.push_back(c_);
      c_ = 0;
    }

    const list<int>& historia() const {
      return historia_;
    }

    int contador() {
      return c_;
    }

  private:
    int c_;
    list<int> historia_;
};
```

```
int main() {
  Cronometro c;
  c.sumar();
  const Cronometro& cr = c;
  const list<int>& l =
  ↪   cr.historia();
}
```

Classes

```
class Cronometro {
  public:
    Cronometro() : c_(0), historia_() {}

    void sumar() {
      c_++;
    }

    void resetear() {
      historia_.push_back(c_);
      c_ = 0;
    }

    const list<int>& historia() const {
      cout << "Version const" << endl;
      return historia_;
    }

    const list<int>& historia() {
      cout << "Version no-const" << endl;
      return historia_;
    }

    int contador() {
      return c_;
    }

  private:
    int c_;
    list<int> historia_;
};
```

```
int main() {
  Cronometro c;
  c.sumar();
  list<int>& l =
  ↪   c.historia(); //
  ↪   "Version no-const"
  const Cronometro& cr = c;
  const list<int>& l =
  ↪   cr.historia(); //
  ↪   "Version const"
}
```

Lista::iesimo

```
template<class T>
class Lista<T> {
  public:
    ...

    /**
     * Devuelve el elemento en la i-ésima posición de la Lista.
     * @param i posición del elemento a devolver.
     * @return referencia no modificable
     */
    const T& iesimo(Nat i) const;
    /**
     * Devuelve el elemento en la i-ésima posición de la Lista.
     * @param i posición del elemento a devolver.
     * @return referencia modificable
     */
    T& iesimo(Nat i);

    ...

};
```