Introducción a C++

Miguel Raggi

Algoritmos
ENES
UNAM

29 de enero de 2019

Índice:

Índice:

Propiedades de C++

Como lenguaje de programación, se clasifica así:

Propiedades de C++

Como lenguaje de programación, se clasifica así:

- Es compilado.

Propiedades de C++

Como lenguaje de programación, se clasifica así:

- Es compilado.
- Es de propósito general: Puedes hacer cualquier tipo de programa.

Propiedades de C++

Como lenguaje de programación, se clasifica así:

- Es compilado.
- Es de propósito general: Puedes hacer cualquier tipo de programa.
- Es de tipos estáticos (statically typed): Tienes que declarar qué tipo de variable es cada una.

Propiedades de C++

Como lenguaje de programación, se clasifica así:

- Es compilado.
- Es de propósito general: Puedes hacer cualquier tipo de programa.
- Es de tipos estáticos (statically typed): Tienes que declarar qué tipo de variable es cada una.
- Es orientado a objetos: Puedes definir nuevos "tipos" y definirle sus propiedades, etc.

Propiedades de C++

Como lenguaje de programación, se clasifica así:

- Es compilado.
- Es de propósito general: Puedes hacer cualquier tipo de programa.
- Es de tipos estáticos (statically typed): Tienes que declarar qué tipo de variable es cada una.
- Es orientado a objetos: Puedes definir nuevos "tipos" y definirle sus propiedades, etc.
- Es free form: Ignora espacios en blanco.

Compilar vs Interpretar

Compilar vs Interpretar

Compilar: La computadora lee los comandos que escribiste y los convierte a lenguaje de máquina, creando un archivo ejecutable que contiene las instrucciones que nosotros le dijimos que tuviera.

Compilar vs Interpretar

Compilar: La computadora lee los comandos que escribiste y los convierte a lenguaje de máquina, creando un archivo ejecutable que contiene las instrucciones que nosotros le dijimos que tuviera.

Interpretar: La computadora lee los comandos que escribiste y los ejecuta mientras los va leyendo.

Compilar vs Interpretar

Compilar: La computadora lee los comandos que escribiste y los convierte a lenguaje de máquina, creando un archivo ejecutable que contiene las instrucciones que nosotros le dijimos que tuviera.

Interpretar: La computadora lee los comandos que escribiste y los ejecuta mientras los va leyendo.

- Lenguajes Compilados: C/C++, C#, Haskell, Pascal, Fortran, Perl, etc.

Compilar vs Interpretar

Compilar: La computadora lee los comandos que escribiste y los convierte a lenguaje de máquina, creando un archivo ejecutable que contiene las instrucciones que nosotros le dijimos que tuviera.

Interpretar: La computadora lee los comandos que escribiste y los ejecuta mientras los va leyendo.

- Lenguajes Compilados: C/C++, C#, Haskell, Pascal, Fortran, Perl, etc.
- Lenguajes Interpretados: Python, Ruby, R, HTML, javascript, etc.

Compilar vs Interpretar

Compilar: La computadora lee los comandos que escribiste y los convierte a lenguaje de máquina, creando un archivo ejecutable que contiene las instrucciones que nosotros le dijimos que tuviera.

Interpretar: La computadora lee los comandos que escribiste y los ejecuta mientras los va leyendo.

- Lenguajes Compilados: C/C++, C#, Haskell, Pascal, Fortran, Perl, etc.
- Lenguajes Interpretados: Python, Ruby, R, HTML, javascript, etc.
- Lenguajes Mixtos: Java (caso especial), Php, etc.

Ventajas y Desventajas

Ventajas de compilar:

Ventajas y Desventajas

Ventajas de compilar:

- El programa ejecutado es mucho, mucho más rápido cuando está compilado por varias razones diferentes:

Ventajas y Desventajas

Ventajas de compilar:

- El programa ejecutado es mucho, mucho más rápido cuando está compilado por varias razones diferentes:
  1. No tiene que leer el texto que tú escribiste, las instrucciones van directo al procesador.

Ventajas y Desventajas

Ventajas de compilar:

- El programa ejecutado es mucho, mucho más rápido cuando está compilado por varias razones diferentes:
  1. No tiene que leer el texto que tú escribiste, las instrucciones van directo al procesador.
  2. El compilador tiene un optimizador incluído.

Ventajas y Desventajas

Ventajas de compilar:

- El programa ejecutado es mucho, mucho más rápido cuando está compilado por varias razones diferentes:
  1. No tiene que leer el texto que tú escribiste, las instrucciones van directo al procesador.
  2. El compilador tiene un optimizador incluído.
- Se puede distribuir el programa sin necesidad de que la otra persona tenga un compilador.

Ventajas y Desventajas

Ventajas de compilar:

- El programa ejecutado es mucho, mucho más rápido cuando está compilado por varias razones diferentes:
  1. No tiene que leer el texto que tú escribiste, las instrucciones van directo al procesador.
  2. El compilador tiene un optimizador incluído.
- Se puede distribuir el programa sin necesidad de que la otra persona tenga un compilador.

Ventajas de interpretar:

Ventajas y Desventajas

Ventajas de compilar:

- El programa ejecutado es mucho, mucho más rápido cuando está compilado por varias razones diferentes:
  1. No tiene que leer el texto que tú escribiste, las instrucciones van directo al procesador.
  2. El compilador tiene un optimizador incluído.
- Se puede distribuir el programa sin necesidad de que la otra persona tenga un compilador.

Ventajas de interpretar:

- El programa compilado no necesariamente funciona igual en todas las máquinas.

Ventajas y Desventajas

Ventajas de compilar:

- El programa ejecutado es mucho, mucho más rápido cuando está compilado por varias razones diferentes:
  1. No tiene que leer el texto que tú escribiste, las instrucciones van directo al procesador.
  2. El compilador tiene un optimizador incluído.
- Se puede distribuir el programa sin necesidad de que la otra persona tenga un compilador.

Ventajas de interpretar:

- El programa compilado no necesariamente funciona igual en todas las máquinas.
- Puedes ejecutar pedazos del programa, la programación se vuelve "interactiva", en el sentido de que puedes ver el resultado de tus comandos inmediatamente.

Ventajas y Desventajas

Ventajas de compilar:

- El programa ejecutado es mucho, mucho más rápido cuando está compilado por varias razones diferentes:
  1. No tiene que leer el texto que tú escribiste, las instrucciones van directo al procesador.
  2. El compilador tiene un optimizador incluído.
- Se puede distribuir el programa sin necesidad de que la otra persona tenga un compilador.

Ventajas de interpretar:

- El programa compilado no necesariamente funciona igual en todas las máquinas.
- Puedes ejecutar pedazos del programa, la programación se vuelve "interactiva", en el sentido de que puedes ver el resultado de tus comandos inmediatamente.
- Para pasarte un programa, a fuerzas te deben dar el código fuente!

C vs C++

- En 1973 se creó el lenguaje C en Bell Labs por Dennis Ritchie.

C vs C++

- En 1973 se creó el lenguaje C en Bell Labs por Dennis Ritchie.
- Desde 1979 se comenzó a añadir cosas, como clases, funciones virtuales, excepciones, etc.

C vs C++

- En 1973 se creó el lenguaje C en Bell Labs por Dennis Ritchie.
- Desde 1979 se comenzó a añadir cosas, como clases, funciones virtuales, excepciones, etc.
- En 1983 se le nombró C++, por Bjarne Stroustrup.

C vs C++

- En 1973 se creó el lenguaje C en Bell Labs por Dennis Ritchie.
- Desde 1979 se comenzó a añadir cosas, como clases, funciones virtuales, excepciones, etc.
- En 1983 se le nombró C++, por Bjarne Stroustrup.
- En 1998 finalmente se ratificó el primer estándar de C++.

C vs C++

- En 1973 se creó el lenguaje C en Bell Labs por Dennis Ritchie.
- Desde 1979 se comenzó a añadir cosas, como clases, funciones virtuales, excepciones, etc.
- En 1983 se le nombró C++, por Bjarne Stroustrup.
- En 1998 finalmente se ratificó el primer estándar de C++.
- En 2003 se añadieron algunas cosas.

C vs C++

- En 1973 se creó el lenguaje C en Bell Labs por Dennis Ritchie.
- Desde 1979 se comenzó a añadir cosas, como clases, funciones virtuales, excepciones, etc.
- En 1983 se le nombró C++, por Bjarne Stroustrup.
- En 1998 finalmente se ratificó el primer estándar de C++.
- En 2003 se añadieron algunas cosas.
- En 2011 se le añadieron un montón de cosas nuevas a C++, como lambdas, movimiento de objetos, auto, apuntadores inteligentes, etc. que hacen del lenguaje más moderno, más rápido y más fácil de utilizar.

C vs C++

- En 1973 se creó el lenguaje C en Bell Labs por Dennis Ritchie.
- Desde 1979 se comenzó a añadir cosas, como clases, funciones virtuales, excepciones, etc.
- En 1983 se le nombró C++, por Bjarne Stroustrup.
- En 1998 finalmente se ratificó el primer estándar de C++.
- En 2003 se añadieron algunas cosas.
- En 2011 se le añadieron un montón de cosas nuevas a C++, como lambdas, movimiento de objetos, auto, apuntadores inteligentes, etc. que hacen del lenguaje más moderno, más rápido y más fácil de utilizar.
- En 2014 se añaden algunas cosas más (lambdas con auto y cosas así).

C vs C++

- En 1973 se creó el lenguaje C en Bell Labs por Dennis Ritchie.
- Desde 1979 se comenzó a añadir cosas, como clases, funciones virtuales, excepciones, etc.
- En 1983 se le nombró C++, por Bjarne Stroustrup.
- En 1998 finalmente se ratificó el primer estándar de C++.
- En 2003 se añadieron algunas cosas.
- En 2011 se le añadieron un montón de cosas nuevas a C++, como lambdas, movimiento de objetos, auto, apuntadores inteligentes, etc. que hacen del lenguaje más moderno, más rápido y más fácil de utilizar.
- En 2014 se añaden algunas cosas más (lambdas con auto y cosas así).
- La última versión es C++17, que tiene varias nuevas cosas, aunque los compiladores todavía no funcionan bien.

Propiedades de C++

Además, tiene las siguientes propiedades “subjetivas”:

Propiedades de C++

Además, tiene las siguientes propiedades “subjetivas”:

- Es muy, muy rápido. De los lenguajes populares, es el más rápido.

Propiedades de C++

Además, tiene las siguientes propiedades "subjetivas":

- Es muy, muy rápido. De los lenguajes populares, es el más rápido.
- Es muy popular. La mayoría del software que corres en tu computadora está hecho en C++.

Propiedades de C++

Además, tiene las siguientes propiedades "subjetivas":

- Es muy, muy rápido. De los lenguajes populares, es el más rápido.
- Es muy popular. La mayoría del software que corres en tu computadora está hecho en C++.
- La biblioteca estándar que viene con C++ está muy bien diseñada.

Propiedades de C++

Además, tiene las siguientes propiedades "subjetivas":

- Es muy, muy rápido. De los lenguajes populares, es el más rápido.
- Es muy popular. La mayoría del software que corres en tu computadora está hecho en C++.
- La biblioteca estándar que viene con C++ está muy bien diseñada.
- Está diseñado para proyectos grandes, en donde mucha gente está trabajando en diferentes partes del código simultáneamente.

Propiedades de C++

Además, tiene las siguientes propiedades "subjetivas":

- Es muy, muy rápido. De los lenguajes populares, es el más rápido.
- Es muy popular. La mayoría del software que corres en tu computadora está hecho en C++.
- La biblioteca estándar que viene con C++ está muy bien diseñada.
- Está diseñado para proyectos grandes, en donde mucha gente está trabajando en diferentes partes del código simultáneamente.
- Es difícil de aprender.

Propiedades de C++

Además, tiene las siguientes propiedades "subjetivas":

- Es muy, muy rápido. De los lenguajes populares, es el más rápido.
- Es muy popular. La mayoría del software que corres en tu computadora está hecho en C++.
- La biblioteca estándar que viene con C++ está muy bien diseñada.
- Está diseñado para proyectos grandes, en donde mucha gente está trabajando en diferentes partes del código simultáneamente.
- Es difícil de aprender.
- Hay muchísimas bibliotecas libres que puedes utilizar si utilizas C++.

¿Cuándo utilizar cada uno?

En realidad muchos proyectos grandes utilizan 2 o más lenguajes:

¿Cuándo utilizar cada uno?

En realidad muchos proyectos grandes utilizan 2 o más lenguajes:

- Para la parte que es pesada computacionalmente utilizan algún lenguaje compilado como C++ y lo trabajan los meros expertos.

¿Cuándo utilizar cada uno?

En realidad muchos proyectos grandes utilizan 2 o más lenguajes:

- Para la parte que es pesada computacionalmente utilizan algún lenguaje compilado como C++ y lo trabajan los meros expertos.
- Para la parte que es ligera y que requiere estar modificando constantemente, utilizan un lenguaje más ligero como python o lua, y, como es más fácil, pueden trabajarlo los no tan expertos.

¿Cuándo utilizar cada uno?

En realidad muchos proyectos grandes utilizan 2 o más lenguajes:

- Para la parte que es pesada computacionalmente utilizan algún lenguaje compilado como C++ y lo trabajan los meros expertos.
- Para la parte que es ligera y que requiere estar modificando constantemente, utilizan un lenguaje más ligero como python o lua, y, como es más fácil, pueden trabajarlo los no tan expertos.
- Si trabajas para una compañía o con algún grupo de trabajo, muchas veces tú no decides qué lenguaje utilizar, te tienes que adaptar al que se está utilizando para el proyecto.

¿Cuándo utilizar cada uno?

En realidad muchos proyectos grandes utilizan 2 o más lenguajes:

- Para la parte que es pesada computacionalmente utilizan algún lenguaje compilado como C++ y lo trabajan los meros expertos.
- Para la parte que es ligera y que requiere estar modificando constantemente, utilizan un lenguaje más ligero como python o lua, y, como es más fácil, pueden trabajarlo los no tan expertos.
- Si trabajas para una compañía o con algún grupo de trabajo, muchas veces tú no decides qué lenguaje utilizar, te tienes que adaptar al que se está utilizando para el proyecto.
- Les enseñaremos C++ en este curso porque nos permite entender más profundamente lo que está ocurriendo: python esconde muchos detalles de ti.

¿Cuándo utilizar cada uno?

En realidad muchos proyectos grandes utilizan 2 o más lenguajes:

- Para la parte que es pesada computacionalmente utilizan algún lenguaje compilado como C++ y lo trabajan los meros expertos.
- Para la parte que es ligera y que requiere estar modificando constantemente, utilizan un lenguaje más ligero como python o lua, y, como es más fácil, pueden trabajarlo los no tan expertos.
- Si trabajas para una compañía o con algún grupo de trabajo, muchas veces tú no decides qué lenguaje utilizar, te tienes que adaptar al que se está utilizando para el proyecto.
- Les enseñaremos C++ en este curso porque nos permite entender más profundamente lo que está ocurriendo: python esconde muchos detalles de ti.
- Una vez que aprendes un lenguaje pesado bien, todos los demás son fáciles de aprender, pero seguramente a lo largo de su vida tendrán que aprender por si mismos muchos otros lenguajes.

De tarea para siempre: Tutoriales de C++ hasta que lo aprendas bien. Después, tutoriales de algún otro que te guste (D, Haskell, Kotlin, javascript, Julia, etc.)

Compilación

Ya vimos que C++ es un lenguaje compilado. Pero hay varios compiladores diferentes. Algunos ejemplos:

Compilación

Ya vimos que C++ es un lenguaje compilado. Pero hay varios compiladores diferentes. Algunos ejemplos:

- GNU g++: Parte de la GNU Compiler Collection (gcc). Es muy popular, de fuente abierta, muy viejo, y tiene versiones para Linux, mac, Windows, etc.

Compilación

Ya vimos que C++ es un lenguaje compilado. Pero hay varios compiladores diferentes. Algunos ejemplos:

- GNU g++: Parte de la GNU Compiler Collection (gcc). Es muy popular, de fuente abierta, muy viejo, y tiene versiones para Linux, mac, Windows, etc.
- LLVM clang: También de fuente abierta, es más moderno que gcc y está diseñado para reemplazarlo. Todavía gcc le gana en velocidad de ejecución a clang, pero clang compila mucho más rápido.

Compilación

Ya vimos que C++ es un lenguaje compilado. Pero hay varios compiladores diferentes. Algunos ejemplos:

- GNU g++: Parte de la GNU Compiler Collection (gcc). Es muy popular, de fuente abierta, muy viejo, y tiene versiones para Linux, mac, Windows, etc.
- LLVM clang: También de fuente abierta, es más moderno que gcc y está diseñado para reemplazarlo. Todavía gcc le gana en velocidad de ejecución a clang, pero clang compila mucho más rápido.
- Intel C++ compiler: Es rápido porque utiliza propiedades de los procesadores Intel para correr mejor en ellos. Cuesta $$$.

Compilación

Ya vimos que C++ es un lenguaje compilado. Pero hay varios compiladores diferentes. Algunos ejemplos:

- GNU g++: Parte de la GNU Compiler Collection (gcc). Es muy popular, de fuente abierta, muy viejo, y tiene versiones para Linux, mac, Windows, etc.
- LLVM clang: También de fuente abierta, es más moderno que gcc y está diseñado para reemplazarlo. Todavía gcc le gana en velocidad de ejecución a clang, pero clang compila mucho más rápido.
- Intel C++ compiler: Es rápido porque utiliza propiedades de los procesadores Intel para correr mejor en ellos. Cuesta $$$.
- Visual C++: Es el de microsoft. Cuesta $$$, y sólo funciona en Windows. Es popular porque viene pegado del editor Visual Studio, que es bastante bueno, pero sólo funciona en Windows. (no usar)

Compilador vs IDE

- El compilador es el programa que convierte tus instrucciones en lenguaje de máquina.

Compilador vs IDE

- El compilador es el programa que convierte tus instrucciones en lenguaje de máquina.
- Sin embargo, uno como programador no tiene tanta interacción con el compilador.

Compilador vs IDE

- El compilador es el programa que convierte tus instrucciones en lenguaje de máquina.
- Sin embargo, uno como programador no tiene tanta interacción con el compilador.
- Cuando haces un programa en C++, en teoría, podrías simplemente escribir todas las instrucciones en el bloc de notas o cualquier otro editor de texto, y luego decirle al compilador que lo compile con comandos así:

```
g++ miarchivo.cpp -o miprograma
```

Compilador vs IDE

- El compilador es el programa que convierte tus instrucciones en lenguaje de máquina.
- Sin embargo, uno como programador no tiene tanta interacción con el compilador.
- Cuando haces un programa en C++, en teoría, podrías simplemente escribir todas las instrucciones en el bloc de notas o cualquier otro editor de texto, y luego decirle al compilador que lo compile con comandos así:

```
g++ miarchivo.cpp -o miprograma
```

- Pero claro, cuando tienes 30 archivos y opciones complicadas, no quieres estar escribiendo esto cada vez. Sería mejor si con un sólo botón que aprietes, se compile todo.

Compilador vs IDE

- El compilador es el programa que convierte tus instrucciones en lenguaje de máquina.
- Sin embargo, uno como programador no tiene tanta interacción con el compilador.
- Cuando haces un programa en C++, en teoría, podrías simplemente escribir todas las instrucciones en el bloc de notas o cualquier otro editor de texto, y luego decirle al compilador que lo compile con comandos así:

```
g++ miarchivo.cpp -o miprograma
```

- Pero claro, cuando tienes 30 archivos y opciones complicadas, no quieres estar escribiendo esto cada vez. Sería mejor si con un sólo botón que aprietes, se compile todo.
- Por eso utilizamos lo que se llama IDE (Integrated Development Environment).

¿Qué es una IDE?

IDE = Intergrated Development Environment

¿Qué es una IDE?

IDE = Intergrated Development Environment

- Una IDE es un editor de texto diseñado para facilitar la programación.

¿Qué es una IDE?

IDE = Intergrated Development Environment

- Una IDE es un editor de texto diseñado para facilitar la programación.
- Por lo menos debes poder editar archivos y compilar, pero casi siempre traen cosas extra que te ayudan a programar.

¿Qué es una IDE?

IDE = Intergrated Development Environment

- Una IDE es un editor de texto diseñado para facilitar la programación.
- Por lo menos debes poder editar archivos y compilar, pero casi siempre traen cosas extra que te ayudan a programar.
- Por ejemplo, por lo menos una IDE debe ponerle colores al texto, para ayudarte a leer el programa.

¿Qué es una IDE?

IDE = Intergrated Development Environment

- Una IDE es un editor de texto diseñado para facilitar la programación.
- Por lo menos debes poder editar archivos y compilar, pero casi siempre traen cosas extra que te ayudan a programar.
- Por ejemplo, por lo menos una IDE debe ponerle colores al texto, para ayudarte a leer el programa.
- Los IDEs buenos traen completación de palabras y de comandos, te marcan los errores, puedes ver donde definiste ciertas variables, modificar las opciones de compilación, etc, etc.

¿Qué es una IDE?

IDE = Intergrated Development Environment

- Una IDE es un editor de texto diseñado para facilitar la programación.
- Por lo menos debes poder editar archivos y compilar, pero casi siempre traen cosas extra que te ayudan a programar.
- Por ejemplo, por lo menos una IDE debe ponerle colores al texto, para ayudarte a leer el programa.
- Los IDEs buenos traen completación de palabras y de comandos, te marcan los errores, puedes ver donde definiste ciertas variables, modificar las opciones de compilación, etc, etc.
- A veces puedes utilizar el IDE y el compilador que quieras, pero varios IDEs están hechos para cierto compilador específico.

¿Qué IDEs hay?

Las más conocidas para C++ son:

¿Qué IDEs hay?

Las más conocidas para C++ son:

- Visual Studio: Es de Microsoft, sólo funciona en Windows. Cuesta $$$.

¿Qué IDEs hay?

Las más conocidas para C++ son:

- Visual Studio: Es de Microsoft, sólo funciona en Windows. Cuesta $$$.
- Xcode: Es de Apple, sólo funciona en Mac, gratis.

¿Qué IDEs hay?

Las más conocidas para C++ son:

- Visual Studio: Es de Microsoft, sólo funciona en Windows. Cuesta $$$.
- Xcode: Es de Apple, sólo funciona en Mac, gratis.
- Code::Blocks: Es de fuente abierta y gratis. Es muy fácil de utilizar, y funciona en Windows, Linux, Mac, etc.

¿Qué IDEs hay?

Las más conocidas para C++ son:

- Visual Studio: Es de Microsoft, sólo funciona en Windows. Cuesta $$$.
- Xcode: Es de Apple, sólo funciona en Mac, gratis.
- Code::Blocks: Es de fuente abierta y gratis. Es muy fácil de utilizar, y funciona en Windows, Linux, Mac, etc.
- Kdevelop: Igual, de fuente abierta y gratis. Es más difícil de utilizar, y sólo funciona en Linux. Es el que yo uso, porque es muy muy bueno.

¿Qué IDEs hay?

Las más conocidas para C++ son:

- Visual Studio: Es de Microsoft, sólo funciona en Windows. Cuesta $$$.
- Xcode: Es de Apple, sólo funciona en Mac, gratis.
- Code::Blocks: Es de fuente abierta y gratis. Es muy fácil de utilizar, y funciona en Windows, Linux, Mac, etc.
- Kdevelop: Igual, de fuente abierta y gratis. Es más difícil de utilizar, y sólo funciona en Linux. Es el que yo uso, porque es muy muy bueno.
- Qtcreator: fuente abierta y gratis, especializado para hacer programas que utilicen Qt, una biblioteca para hacer ventanitas y eso. Linux, Windows, Mac...

¿Qué IDEs hay?

Las más conocidas para C++ son:

- Visual Studio: Es de Microsoft, sólo funciona en Windows. Cuesta $$$.
- Xcode: Es de Apple, sólo funciona en Mac, gratis.
- Code::Blocks: Es de fuente abierta y gratis. Es muy fácil de utilizar, y funciona en Windows, Linux, Mac, etc.
- Kdevelop: Igual, de fuente abierta y gratis. Es más difícil de utilizar, y sólo funciona en Linux. Es el que yo uso, porque es muy muy bueno.
- Qtcreator: fuente abierta y gratis, especializado para hacer programas que utilicen Qt, una biblioteca para hacer ventanitas y eso. Linux, Windows, Mac...
- Emacs o Vim: Editores de texto muy poderosos, que los que lo saben utilizar pueden ser muy rápidos con ellos. Es difícil y antiguo.

IDEs

- Hay varios otros IDEs, y en general es cuestión de gustos.

IDEs

- Hay varios otros IDEs, y en general es cuestión de gustos.
- Si trabajas en una compañía o con un grupo de gente, usualmente tienes que utilizar el que utilizan los demás, tanto en compilador como en IDE.

IDEs

- Hay varios otros IDEs, y en general es cuestión de gustos.
- Si trabajas en una compañía o con un grupo de gente, usualmente tienes que utilizar el que utilizan los demás, tanto en compilador como en IDE.
- Nosotros utilizaremos Code::Blocks porque es fácil para principiantes, popular, gratis, y está en Windows, Mac, Linux, etc.

IDEs

- Hay varios otros IDEs, y en general es cuestión de gustos.
- Si trabajas en una compañía o con un grupo de gente, usualmente tienes que utilizar el que utilizan los demás, tanto en compilador como en IDE.
- Nosotros utilizaremos Code::Blocks porque es fácil para principiantes, popular, gratis, y está en Windows, Mac, Linux, etc.
- También se conecta muy fácilmente con GCC o clang, y de hecho puedes bajar el Code::Blocks ya con el compilador MinGW (g++ para Windows) incluido.

IDEs

- Hay varios otros IDEs, y en general es cuestión de gustos.
- Si trabajas en una compañía o con un grupo de gente, usualmente tienes que utilizar el que utilizan los demás, tanto en compilador como en IDE.
- Nosotros utilizaremos Code::Blocks porque es fácil para principiantes, popular, gratis, y está en Windows, Mac, Linux, etc.
- También se conecta muy fácilmente con GCC o clang, y de hecho puedes bajar el Code::Blocks ya con el compilador MinGW (g++ para Windows) incluido.
- Como siempre, en Linux es más fácil programar, porque es muy fácil instalar bibliotecas que quieres.

A programar!

¿Qué hace el compilador de C++?

Cuando le pedimos al compilador que compile nuestro código, hace 3 cosas:

¿Qué hace el compilador de C++?

Cuando le pedimos al compilador que compile nuestro código, hace 3 cosas:

- Preprocesador: Son instrucciones que no convierte directamente en código, sino que modifica el código de cierta manera: Por ejemplo, mete tal archivo aquí, ignora este pedazo de código si estás en linux, etc.

¿Qué hace el compilador de C++?

Cuando le pedimos al compilador que compile nuestro código, hace 3 cosas:

- Preprocesador: Son instrucciones que no convierte directamente en código, sino que modifica el código de cierta manera: Por ejemplo, mete tal archivo aquí, ignora este pedazo de código si estás en linux, etc.
- Compilador: Convierte el código en lenguaje de máquina. También, si lo pides, corre un optimizador.

¿Qué hace el compilador de C++?

Cuando le pedimos al compilador que compile nuestro código, hace 3 cosas:

- Preprocesador: Son instrucciones que no convierte directamente en código, sino que modifica el código de cierta manera: Por ejemplo, mete tal archivo aquí, ignora este pedazo de código si estás en linux, etc.
- Compilador: Convierte el código en lenguaje de máquina. También, si lo pides, corre un optimizador.
- Ligador (linker): Liga las bibliotecas extra que le hayas pedido, por ejemplo, openGL, SDL, SFML, libnoise, boost, etc.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores. Además, hay 0 personas en el mundo que pueden escribir programas largos que compilen y corran bien a la primera.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores. Además, hay 0 personas en el mundo que pueden escribir programas largos que compilen y corran bien a la primera.

- Todos esos maestros que dicen que "lo que importa es el resultado" y que si te equivocas en una cuenta te ponen mal todo nunca han programado.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores. Además, hay 0 personas en el mundo que pueden escribir programas largos que compilen y corran bien a la primera.

- Todos esos maestros que dicen que "lo que importa es el resultado" y que si te equivocas en una cuenta te ponen mal todo nunca han programado.
- No es posible no equivocarse en un programa medianamente grande.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores. Además, hay 0 personas en el mundo que pueden escribir programas largos que compilen y corran bien a la primera.

- Todos esos maestros que dicen que "lo que importa es el resultado" y que si te equivocas en una cuenta te ponen mal todo nunca han programado.
- No es posible no equivocarse en un programa medianamente grande.
- Entonces todo lo debes estar probando constantemente.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores. Además, hay 0 personas en el mundo que pueden escribir programas largos que compilen y corran bien a la primera.

- Todos esos maestros que dicen que "lo que importa es el resultado" y que si te equivocas en una cuenta te ponen mal todo nunca han programado.
- No es posible no equivocarse en un programa medianamente grande.
- Entonces todo lo debes estar probando constantemente.
- No se deben asustar cuando tratas de compilar y te salen 1000 errores. Es perfectamente normal.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores. Además, hay 0 personas en el mundo que pueden escribir programas largos que compilen y corran bien a la primera.

- Todos esos maestros que dicen que "lo que importa es el resultado" y que si te equivocas en una cuenta te ponen mal todo nunca han programado.
- No es posible no equivocarse en un programa medianamente grande.
- Entonces todo lo debes estar probando constantemente.
- No se deben asustar cuando tratas de compilar y te salen 1000 errores. Es perfectamente normal.
- De hecho, yo me sorprendo mucho cuando escribo algo, lo compilo, y no da errores.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores. Además, hay 0 personas en el mundo que pueden escribir programas largos que compilen y corran bien a la primera.

- Todos esos maestros que dicen que "lo que importa es el resultado" y que si te equivocas en una cuenta te ponen mal todo nunca han programado.
- No es posible no equivocarse en un programa medianamente grande.
- Entonces todo lo debes estar probando constantemente.
- No se deben asustar cuando tratas de compilar y te salen 1000 errores. Es perfectamente normal.
- De hecho, yo me sorprendo mucho cuando escribo algo, lo compilo, y no da errores. Usualmente después al correr el programa se traba.

Siempre habrá errores

Teorema

Todo programa de computación contiene errores. Además, hay 0 personas en el mundo que pueden escribir programas largos que compilen y corran bien a la primera.

- Todos esos maestros que dicen que "lo que importa es el resultado" y que si te equivocas en una cuenta te ponen mal todo nunca han programado.
- No es posible no equivocarse en un programa medianamente grande.
- Entonces todo lo debes estar probando constantemente.
- No se deben asustar cuando tratas de compilar y te salen 1000 errores. Es perfectamente normal.
- De hecho, yo me sorprendo mucho cuando escribo algo, lo compilo, y no da errores. Usualmente después al correr el programa se traba.

Tipos de Errores

Hay 2 tipos de errores:

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.
   - Etc.

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.
   - Etc.
2. Errores de ejecución: Difíciles de detectar y corregir. Por ejemplo:

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.
   - Etc.
2. Errores de ejecución: Difíciles de detectar y corregir. Por ejemplo:
   - Cuando corro el programa, se traba inmediatamente.

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.
   - Etc.
2. Errores de ejecución: Difíciles de detectar y corregir. Por ejemplo:
   - Cuando corro el programa, se traba inmediatamente.
   - Si tomo cierta acción en el programa, se sale diciendo "Error de Memoria" o "Segmentation Fault".

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.
   - Etc.
2. Errores de ejecución: Difíciles de detectar y corregir. Por ejemplo:
   - Cuando corro el programa, se traba inmediatamente.
   - Si tomo cierta acción en el programa, se sale diciendo "Error de Memoria" o "Segmentation Fault".
   - Etc.

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.
   - Etc.
2. Errores de ejecución: Difíciles de detectar y corregir. Por ejemplo:
   - Cuando corro el programa, se traba inmediatamente.
   - Si tomo cierta acción en el programa, se sale diciendo "Error de Memoria" o "Segmentation Fault".
   - Etc.

Uno entiende más a C++ si entiende que mucho del lenguaje está hecho para convertir errores de ejecución en errores de compilación:

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.
   - Etc.
2. Errores de ejecución: Difíciles de detectar y corregir. Por ejemplo:
   - Cuando corro el programa, se traba inmediatamente.
   - Si tomo cierta acción en el programa, se sale diciendo "Error de Memoria" o "Segmentation Fault".
   - Etc.

Uno entiende más a C++ si entiende que mucho del lenguaje está hecho para convertir errores de ejecución en errores de compilación: A veces parece que te estás poniendo trabas a ti mismo...

Tipos de Errores

Hay 2 tipos de errores:

1. Errores de compilación: Fáciles de detectar y de corregir. Por ejemplo:
   - Se me olvidó un ;
   - Puse un nombre de variable diferente antes y después.
   - Traté de modificar algo que había declarado constante.
   - Etc.
2. Errores de ejecución: Difíciles de detectar y corregir. Por ejemplo:
   - Cuando corro el programa, se traba inmediatamente.
   - Si tomo cierta acción en el programa, se sale diciendo "Error de Memoria" o "Segmentation Fault".
   - Etc.

Uno entiende más a C++ si entiende que mucho del lenguaje está hecho para convertir errores de ejecución en errores de compilación: A veces parece que te estás poniendo trabas a ti mismo... ¡pero eso es exactamente lo que quieres!

Debugging

- Ya que tienes un error de ejecución que no sabes qué pasa, entonces entran los debuggers (*depuradores*).

Debugging

- Ya que tienes un error de ejecución que no sabes qué pasa, entonces entran los debuggers (*depuradores*).
- Básicamente, son programas que te permiten ejecutar tu programa paso a paso, y revisar cuánto vale cada variable, etc.

Debugging

- Ya que tienes un error de ejecución que no sabes qué pasa, entonces entran los debuggers (*depuradores*).
- Básicamente, son programas que te permiten ejecutar tu programa paso a paso, y revisar cuánto vale cada variable, etc.
- Esto con el fin de darte cuenta exactamente en donde tienes el error.

Debugging

- Ya que tienes un error de ejecución que no sabes qué pasa, entonces entran los debuggers (*depuradores*).
- Básicamente, son programas que te permiten ejecutar tu programa paso a paso, y revisar cuánto vale cada variable, etc.
- Esto con el fin de darte cuenta exactamente en donde tienes el error.
- Mucha de tu vida de programador la pasas haciendo exactamente esto.

Debugging

- Ya que tienes un error de ejecución que no sabes qué pasa, entonces entran los debuggers (*depuradores*).
- Básicamente, son programas que te permiten ejecutar tu programa paso a paso, y revisar cuánto vale cada variable, etc.
- Esto con el fin de darte cuenta exactamente en donde tienes el error.
- Mucha de tu vida de programador la pasas haciendo exactamente esto.
- Obviamente veremos esto mucho más a fondo.

Debugging

- Ya que tienes un error de ejecución que no sabes qué pasa, entonces entran los debuggers (*depuradores*).
- Básicamente, son programas que te permiten ejecutar tu programa paso a paso, y revisar cuánto vale cada variable, etc.
- Esto con el fin de darte cuenta exactamente en donde tienes el error.
- Mucha de tu vida de programador la pasas haciendo exactamente esto.
- Obviamente veremos esto mucho más a fondo.
- La forma más simple de debuggear es ponerle al programa que imprima cosas cada que hace algo y ver donde se traba.