Auxiliar 2: Computación GPU

CMake y GTest

Profesora: Nancy Hitschfeld-Kahler
Auxiliar: Sergio Salinas
Ayudantes: Roberto Carrasco, Pablo Pizarro

CC7515 - Computación en GPU

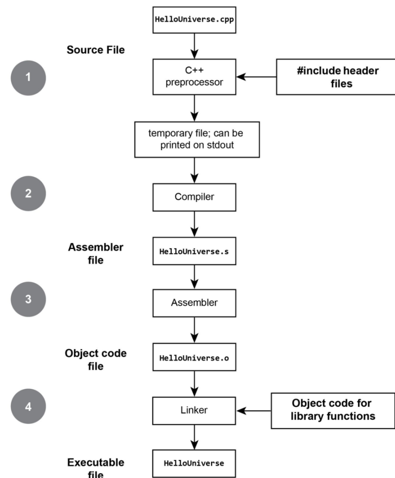
April 12, 2023

1. Introducción a compilación en C++
2. CMake
3. Gtest

Introducción a compilación en C++

Modelo compilación en C++

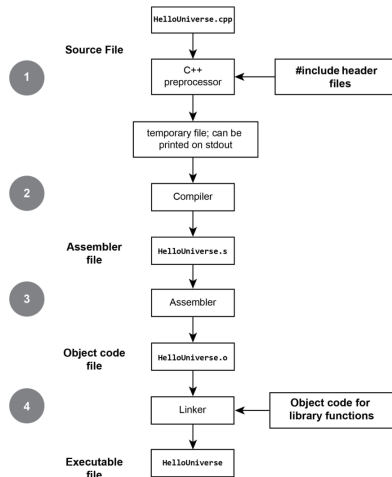
1. El preprocesador es una utilidad que genera el código fuente C que será compilado.
2. El compilador compila el código fuente C en código objeto, generando un conjunto de archivos objeto.
3. El enlazador une los distintos archivos objeto, junto con cualquier biblioteca necesaria, en un archivo ejecutable.



Modelo compilación en C++

1. Preprocesamiento (.cpp + .h -> .ii)
2. Compilación (.ii -> .o/.a) o (.ii -> .so/.dll)
3. Linking (.o + .a -> .out) o (.so + .dll -> .exe)

Hay que decirle al preprocesador dónde buscar headers, y al linker qué librerías incluir en la compilación y también dónde buscarlas.



Preprocesador y linker

- ¿Cómo le decimos al preprocesador dónde buscar archivos? Se añade la opción guión i mayúscula.

MakeFile

```
gcc -Ipath/to/dir/with/headers archivo.cpp
```

- ¿Cómo le decimos al linker qué librerías buscar? Se añade la opción guion l minúscula

MakeFile

```
gcc -lglut -lpthead archivo.cpp
```

- ¿Cómo le decimos al linker donde buscar librerías? Se añade la opción guion l mayúscula

MakeFile

```
gcc -Lpath/to/dir/with/libraries archivo.cpp
```

Estructura típica de una librería en C++

- CMakeLists.txt: archivo de configuración de CMake que define la compilación de la librería y sus dependencias.
- extern/: directorio que contiene las dependencias externas de la librería.
- include/: directorio que contiene los archivos de cabecera de la librería.
- src/: directorio que contiene los archivos fuente de la librería.
- test/: directorio que contiene los archivos de prueba de la librería.
- examples/: directorio que contiene ejemplos de uso de la librería.
- docs/: directorio que contiene la documentación de la librería.

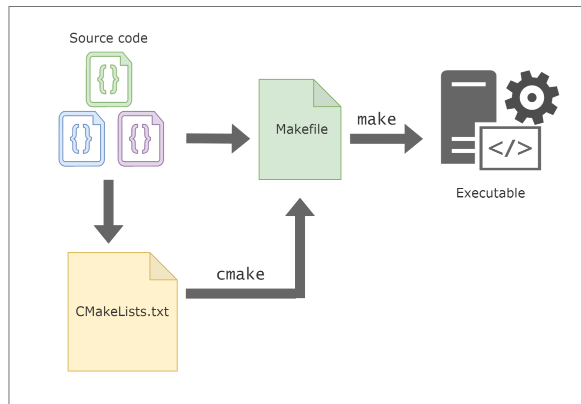
Estructura de un proyecto en CMake

```
mylibrary/
  CMakeLists.txt
  extern/
    some_external_library/
      CMakeLists.txt
      include/
        some_external_library/
          some_external_library.hpp
      src/
        some_external_library.cpp
  include/
    mylibrary/
      mylibrary.hpp
  src/
    mylibrary.cpp
    some_dependency.cpp
  test/
    CMakeLists.txt
    mylibrary_test.cpp
```

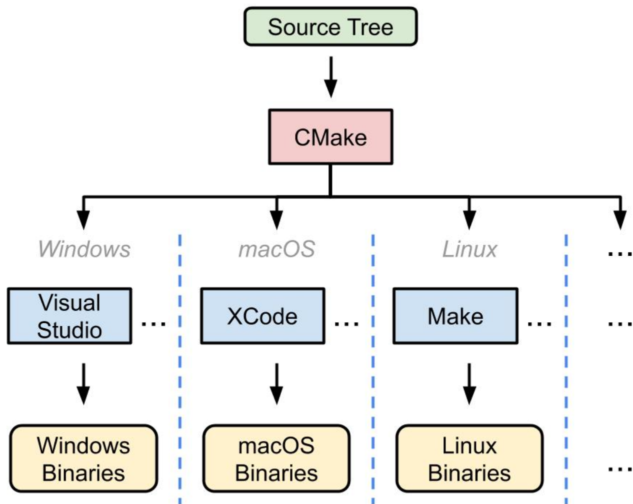
CMake

¿Qué es CMake?

- CMake es una herramienta de código abierto para construir y configurar proyectos de software.
- Automatiza el proceso de construcción de software y genera archivos de configuración para distintos sistemas de construcción.
- Multiplataforma y se puede utilizar en diferentes sistemas operativos.



Ventajas de usar CMake



Ventajas de usar CMake

- **Portabilidad**: CMake se encarga de generar los archivos de configuración necesarios para compilar el software en diferentes plataformas, lo que facilita el proceso de construcción en sistemas operativos como Windows, macOS y Linux.
- **Configuración flexible**: CMake permite configurar opciones de compilación como la elección del compilador, las opciones de optimización y la inclusión de dependencias externas.
- **Modularidad**: CMake permite definir la estructura de un proyecto en módulos, lo que facilita la inclusión y exclusión de partes específicas del proyecto en la construcción.
- **Integración con IDEs**: CMake se integra con varios entornos de desarrollo integrado (IDE), como Visual Studio, Xcode y Qt Creator, lo que facilita el proceso de desarrollo y construcción.
- **Soporte para diferentes lenguajes**: CMake no está limitado al lenguaje de programación C++, sino que también admite otros lenguajes como C, Fortran y Python, entre otros.
- **Compatibilidad con otros sistemas de construcción**: CMake puede generar archivos de configuración para otros sistemas de construcción como Make, Ninja y Visual Studio, lo que hace que sea fácil integrar proyectos CMake en flujos de trabajo existentes.

Ejemplo de CMake

En esta sección, vamos a mostrar un ejemplo sencillo de cómo utilizar CMake para compilar un programa en C++. En este ejemplo, supongamos que queremos compilar un programa que consta de los siguientes archivos:

- `main.cpp`: El archivo fuente principal del programa.
- `util.cpp`: Un archivo fuente que contiene funciones de utilidad.
- `util.h`: El archivo de cabecera correspondiente a 'util.cpp'.

Además, supongamos que queremos generar un ejecutable llamado `my_program`.

Ejemplo de CMake (archivo CMakeLists.txt)

Para compilar nuestro programa utilizando CMake, necesitamos crear un archivo llamado 'CMakeLists.txt'. Aquí está el contenido del archivo para este ejemplo:

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0)
project(my_program)

# Especificar la versión mínima del compilador de C++
set(CMAKE_CXX_STANDARD 17)


# Agregar los archivos fuente al proyecto
add_executable(my_program main.cpp util.cpp)

# Incluir el directorio donde se encuentra el archivo de cabecera
target_include_directories(my_program PRIVATE ${CMAKE_CURRENT_SOURCE_DIR})
```

Ejemplo de CMake (compilación)

Una vez que hemos creado el archivo ‘CMakeLists.txt‘, podemos generar los archivos de configuración y construir el programa ejecutando los siguientes comandos en la terminal:

Terminal

```
mkdir build
cd build
cmake ..
make
```

Estos comandos crearán un directorio de construcción, configurarán el proyecto utilizando el archivo ‘CMakeLists.txt‘ y compilarán el programa para generar el ejecutable `my_program`. El ejecutable se ubicará en la carpeta ‘build/‘.

Gtest

¿Qué es GTest?

- GTest es un framework de pruebas unitarias de código abierto para proyectos en C++.
- Proporciona macros y funciones para definir pruebas, aserciones y casos de prueba.
- Se puede ejecutar en diferentes entornos, incluyendo herramientas de integración continua.
- Genera informes detallados de las pruebas, incluyendo estadísticas de cobertura de código.

google/**googletest**

GoogleTest - Google Testing and Mocking Framework

384 Contributors · 241 Issues · 72 Discussions · 29k Stars · 9k Forks

Usando la librería Gtest de C++

Ejemplo de prueba unitaria con gtest

```cpp
#include <gtest/gtest.h>

TEST(NombreDeLaPrueba, NombreDeLaCondicion) {
  // Código de la prueba aquí.
  // Debe verificar una condición específica.
  // Puedes usar las macros ASSERT_* o EXPECT_*.

    ASSERT_EQ(x.size(), y.size()) << "Vectors x and y are of unequal length";
    for (int i = 0; i < x.size(); ++i) {
        EXPECT_EQ(x[i], y[i]) << "Vectors x and y differ at index " << i;
}
```

Diferencia entre ASSERT y EXPECT en Google Test

En Google Test, ASSERT detiene la ejecución de la prueba inmediatamente si una condición no se cumple, mientras que EXPECT continúa ejecutando la prueba y marca la prueba como fallida al final si la condición no se cumple.

```
TEST(MyTestSuite, MyTestCase) {
    int x = 5;
    ASSERT_EQ(x, 4); // La prueba falla aquí y se detiene.
    EXPECT_EQ(x, 6); // La prueba sigue ejecutando y falla al final.
}
```

En este ejemplo, la primera aserción fallará y detendrá la ejecución de la prueba inmediatamente. La segunda aserción también fallará, pero la prueba seguirá ejecutando hasta el final.

Usando la librería Gtest de C++ con clases

Para usar la librería Gtest de C++ con clases, puedes seguir los siguientes pasos:

```
#include <gtest/gtest.h>

class MiClaseDePrueba : public testing::Test {
 protected:
  // Aquí puedes definir variables y métodos de ayuda para tus pruebas.
};

TEST_F(MiClaseDePrueba, NombreDeLaCondicion) {
  // Código de la prueba aquí.
  // Debe verificar una condición específica.
  // Puedes usar las macros ASSERT_* o EXPECT_*.
}
}

// Más pruebas aquí.

int main(int argc, char **argv) {
  testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

Gtest y Cmake

```
add_executable(mi_ejecutable_de_pruebas
  archivo1.cpp
  archivo2.cpp
  # Agregar más archivos aquí.
)

# Agregar las banderas de compilación y enlazado necesarias.
target_compile_options(mi_ejecutable_de_pruebas PRIVATE -Wall -Wextra -pedantic)
target_link_libraries(mi_ejecutable_de_pruebas gtest gtest_main)

enable_testing()

# Descubrir y ejecutar las pruebas automáticamente.
gtest_discover_tests(mi_ejecutable_de_pruebas)
```

Referencias

- Estructura de proyectos CMake:
  https://cliutils.gitlab.io/modern-cmake/chapters/basics/structure.html
- Assertions en Google Test:
  https://google.github.io/googletest/reference/assertions.html
- Test Fixtures (override en Google Test):
  https://google.github.io/googletest/primer.html
- CGAL con CMake:
  https://doc.cgal.org/latest/Manual/devman_create_and_use_a_cmakelist.html