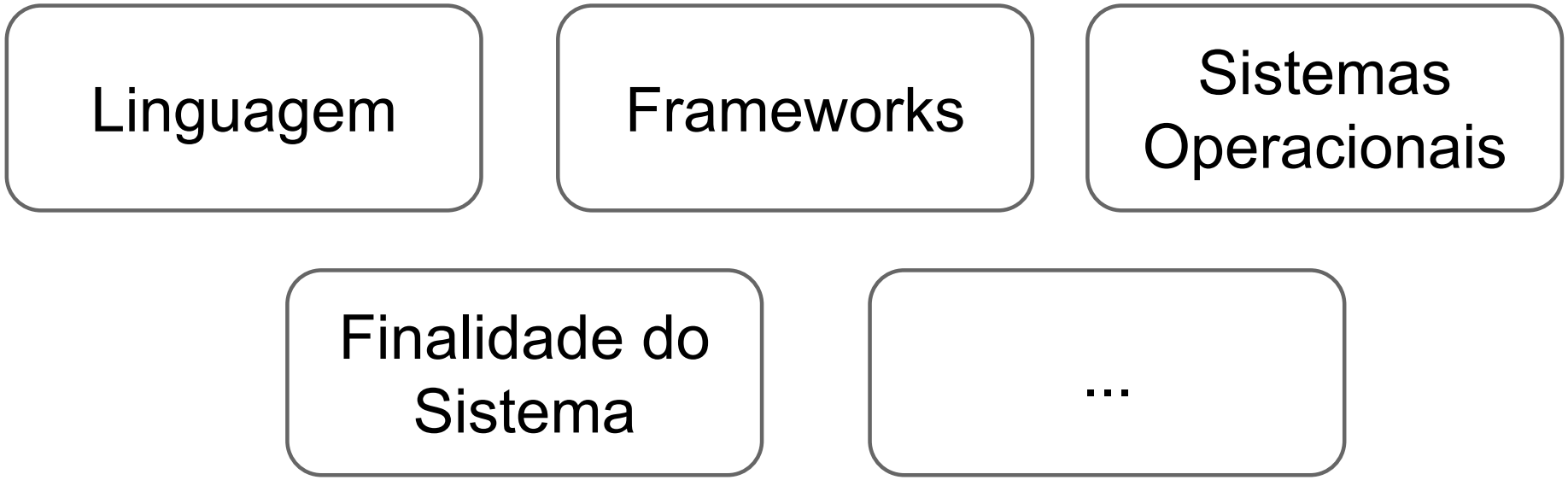
Automação da Build

Prof. Matheus Sousa Faria

Automação da Build

- Automação das tarefas / passos
- Geração de diferentes Builds
  - Desenvolvimento
  - Produção
  - Teste
- Utiliza os dados corretos para cada ambiente
  - Senha
  - Banco
  - Acessos a APIs

Ferramentas



Ferramentas


CMake

- Gerador de Makefiles
- Descreve o processo de Build
- Multiplataforma
- Multi Compilador / Cross Compile
- Conjunto de Ferramentas:
  - Build (CMake)
  - Testa (CTest)
  - Empacota (CPack)
  - Monitora (CDash)
- C / C++
  - C# e CUDA

- Linguagens suportadas
  - ASM
  - ASM-ATT
  - ASM-MASM
  - ASM-NASM
  - C
  - Csharp
  - CUDA
  - CXX
  - Fortran
  - Java
  - RC (Windows Resource Compiler)
  - Swift

- Build Modular
- CMakeLists.txt em cada diretório

```
cmake_minimum_required (VERSION 3.0.2)
project(renderer LANGUAGES CXX)

if(NOT CMAKE_BUILD_TYPE)
    set(CMAKE_BUILD_TYPE Debug)
endif(NOT CMAKE_BUILD_TYPE)

find_package(OpenGL REQUIRED)
find_package(CUDA REQUIRED)

set (PROJECT_INCLUDE_DIRS "${PROJECT_SOURCE_DIR}/include")
set (PROJECT_SRC_DIR     "${PROJECT_SOURCE_DIR}/src")
set (PROJECT_DEPS_DIR    "${PROJECT_SOURCE_DIR}/deps")

add_subdirectory(${PROJECT_DEPS_DIR})
file(GLOB_RECURSE SOURCES "${PROJECT_SRC_DIR}/*.cpp" "${PROJECT_SRC_DIR}/*.c")
file(GLOB_RECURSE SOURCES_CUDA "${PROJECT_SRC_DIR}/*.cu")
```

Básico

O que é necessário em um CMakeLists.txt?

```
# Versão minima do CMake a ser utilizada
cmake_minimum_required(VERSION 3.0.2)

# Definição do nome do projeto
# Descrição das linguagens utilizadas
project(princerescue LANGUAGES CXX)
```

Criação de Variáveis

Criando a variável PROJECT_INCLUDE_DIR

```
set (PROJECT_INCLUDE_DIR "${PROJECT_SOURCE_DIR}/include")
```

Uso do valor de uma variável
${VARIAVEL}

Variável reservada do CMake

Variáveis Reservadas

Vem por padrão para todos os projetos

```
PROJECT_SOURCE_DIR    # Raiz do Projeto
PROJECT_BINARY_DIR    # Diretório de Build
CMAKE_C_COMPILER_ID   # Identificador do Compilador C
CMAKE_CXX_COMPILER_ID # Identificador do Compilador C++
CMAKE_BUILD_TYPE      # Tipo de build escolhida
CMAKE_C_FLAGS         # Flags de Compilação
```

Lista Completa de Variáveis

Configurando Arquivos

Customiza arquivos templates com valores das variáveis

```
set (Tutorial_VERSION_MAJOR 1)
set (Tutorial_VERSION_MINOR 0)
configure_file (
  "${PROJECT_SOURCE_DIR}/TutorialConfig.h.in"
  "${PROJECT_BINARY_DIR}/TutorialConfig.h"
)
```

```
#define Tutorial_VERSION_MAJOR @Tutorial_VERSION_MAJOR@
#define Tutorial_VERSION_MINOR @Tutorial_VERSION_MINOR@
```

TutorialConfig.h.in

Listando arquivos

```
# Listando os arquivos um a um
set(SOURCES “${PROJECT_SRC_DIR}/main.cpp”
            “${PROJECT_SRC_DIR}/graph.cpp”
            “${PROJECT_SRC_DIR}/algebra.cpp”)
# OU
# Listando todos os arquivos automaticamente
file(GLOB_RECURSE SOURCES "${PROJECT_SRC_DIR}/*.cpp"
                          "${PROJECT_SRC_DIR}/*.c")
```

Incluindo os Headers

Incluir os Headers é necessário para o processo de Linkagem

```
# Torna os arquivos .h e .hpp acessiveis aos outros arquivos
# durante a compilação
include_directories("${PROJECT_INCLUDE_DIR}")
```

Adicionando Sub Projetos

Permite a modularização

```
# Adiciona um sub projeto que tenha o CMakeLists.txt na sua raiz
add_subdirectory(${PROJECT_SOURCE_DIR}/libs/graph)
```

```
├── CMakeLists.txt
├── includes
│   ├── pokedex.hpp
├── libs
│   ├── CMakeLists.txt
│   └── graph
│       ├── CMakeLists.txt
│       ├── includes
│       └── sources
└── sources
    ├── main.cpp
    ├── pokedex.cpp
```

Compilação

CMake permite gerar um executável ou biblioteca

```
# Gerando um binário executavel com os *.cpp/*.c
add_executable(NomeDoExecutavel ${SOURCES})


# Gerando um biblioteca estática (.a)
add_library(NomeDaBiblioteca STATIC ${SOURCES})
# Gerando um biblioteca dinâmica (.so, .dll)
add_library(NomeDaBiblioteca SHARED ${SOURCES})
```

Linkagem

Linkando bibliotecas ao seu executavel/biblioteca

```
# Equivalente aos -l… . Exemplo: -lm -lSDL2 -lopencv
target_link_libraries(NomeDoExecutavel
                      Lib1 Lib2 Lib3)
```

Execução

```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ ./programa
```

Gera muitos arquivos temporários

Limpando ambiente

```
$ make clean

OU

$ rm -rf build
```

Seleção do tipo de Build

Customiza a Build para o seu propósito: Debug, Release, Test

```
if(CMAKE_BUILD_TYPE STREQUAL "Debug")
    # Código específioc de Debug
elseif(CMAKE_BUILD_TYPE STREQUAL "Release")
    # Código específico de Release
endif()
```

```
cmake -D CMAKE_BUILD_TYPE=Release ..
```

Comandos customizados

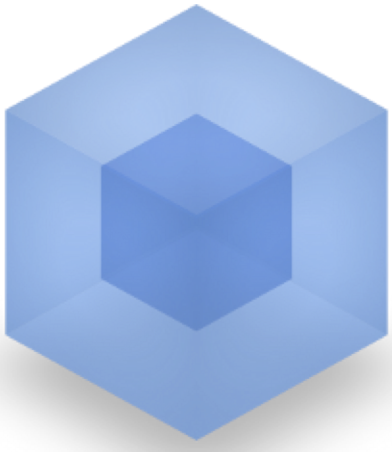
Comandos específicos que não estão por padrão no CMake

```
add_custom_command(TARGET NomeDoExecutavel POST_BUILD
                   COMMAND ${CMAKE_COMMAND} -E copy_directory
                   ${PROJECT_SOURCE_DIR}/assets ${CMAKE_BINARY_DIR}/assets)
```

```
PRE_BUILD   - Antes da build daquele executavel
PRE_LINK    - Após a compilação e antes da linkagem
POST_BUILD  - Após todo o processo
```

Utilizado para arquivos/processos que não fazem parte da compilação.

GRUNT



webpack

MODULE BUNDLER

O que é?

- Task runner / Gerenciador de Tarefas
- Javascript (wat lighting talk)
- Tarefas que auxiliam:
  - Desenvolvimento
  - Deploy
  - Testes
- Automatiza processos
- Disponível no NPM
  - Gerenciador de Pacotes do NodeJS

Dependências (NPM)

- Node Package Manager
- Pacotes em Javascript
- Lista de dependências: package.json
  - Licença
  - Autor
  - Nome do Projeto
  - Descrição
  - Dev, test e deploy
  - scripts

Instalando Dependências

```
$ npm install nome-do-pacote
```

```
$ npm install nome-do-pacote --save-dev
```

Salva como uma dependência de desenvolvimento

package.json
(deps)

$ npm install

```
{
  "name": "project",
  "version": "1.0.0",
  "description": "Gulp and webpack project",
  "main": "index.js",
  "scripts": {
    "gulp": "gulp",
    "test": "exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "browser-sync": "^2.18.13",
    "del": "^3.0.0",
    "gulp": "^3.9.0",
    "gulp-cssnano": "^2.1.2",
    "gulp-sass": "^3.1.0",
    "gulp-uglify": "^3.0.0",
    "gulp-useref": "^3.1.2",
    "run-sequence": "^2.2.0"
  }
}
```

Nome

versão

package.json (scripts)

Executam com o “ambiente” do node

$ npm run gulp

```
{
  "name": "project",
  "version": "1.0.0",
  "description": "Gulp and webpack project",
  "main": "index.js",
  "scripts": {
    "gulp": "gulp",
    "test": "exit 1"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "browser-sync": "^2.18.13",
    "del": "^3.0.0",
    "gulp": "^3.9.0",
    "gulp-cssnano": "^2.1.2",
    "gulp-sass": "^3.1.0",
    "gulp-uglify": "^3.0.0",
    "gulp-useref": "^3.1.2",
    "run-sequence": "^2.2.0"
  }
}
```

Nome

script / commando

Features

- Criação e Execução de tarefas
- Execução assíncrona de tarefas
- Observadores / Watchers

Via plugins:

- Cópia, deleção e criação de arquivos
- Gerenciamento de assets (js, css, html)
- Auto-reloading / carregamento automático de assets
- ...

npm

Estrutura

- gulpfile.js
  - Descrição das tarefas

OU

```
├── tasks
│   ├── index.js
│   ├── test.js
│   └── deploy.js
```

Execução

```
gulp task-name
gulp build
gulp sass
```

Tarefas

```
// “Importando” o módulo Gulp
var gulp = require('gulp');

// Criando uma tarefa
gulp.task('task-name', function() {
  console.log(“minha tarefa!”);
});
```

```
$ gulp task-name
```

Plugins

```
// Importando o plugin
var plugin = require(“gulp-plugin”);

// Utilizando o plugin
gulp.task(“task-name”, function() {
        return gulp.src(“/caminho/arquivos”)
                .pipe(plugin())
                .pipe(gulp.dest(“/caminho/saida”))
})
```

Plugins (exemplo)

```
// Importando o plugin
var sass = require('gulp-sass');

// Utilizando o plugin
gulp.task(“sass”, function() {
        return gulp.src(“app/scss/**/*.scss”)
                .pipe(sass())
                .pipe(gulp.dest(“app/css”))
})
```

Watchers / Observadores

Execução periódica dado um objeto a se observar

```
gulp.task("watch", function() {
        gulp.watch('app/scss/**/*.scss', ['sass'])
})
```

Arquivos para observar

Tarefas para executar

Dependência entre Tarefas

```
gulp.task('task-name', ['task1', 'task2', 'task3'],
function() {
  console.log(“minha tarefa!”);
});
```

Executada antes, de forma assíncrona

Tarefas Sincronas (run-sequence)

```
var runSequence = require('run-sequence');

gulp.task('task-name', function(callback) {
  runSequence('task1',
              'task2',
              ['task2.1', 'task2.2'],
              'task3',
               callback);
});
```

Em paralelo

Tarefa default

Não possui nome

```
gulp.task("default", function() {
        console.log("Tarefa padrao");
})
```

```
$ gulp
```

Perguntas?