Desenvolvimento Aberto

Debugging

Igor dos Santos Montagner (igorsm1@insper.edu.br)

Aulas passadas

- Modelos de contribuição de código
  - Pull Requests, Patches via e-mail, Bug tracker
- Informações importantes de um projeto
- Escolha de um projeto/issue

Hoje

- Compilando seu projeto
- "Regras" de debugging do livro
- Dicas de como se encontrar no projeto

Gerenciamento de projetos de código

Ferramentas

- C/C++ - CMake
- Python - setuptools (pip, conda)
- Java - Graddle, Maven, Ant, ...
- Javascript - npm, bower, ...

Quais problemas básicos são resolvidos?

Gerenciamento de projetos

Duas principais funcionalidades:

1. Definição de quais objetos serão gerados a partir de quais arquivos
    - Ao atualizar um arquivo só reconstrói os objetos que dependem dele
2. Listar as dependências externas do projeto (bibliotecas e binários)
    - Possível instalar dependências com um comando (pip/npm like)

Gerenciamento de projetos pragmático

Só precisamos mesmo de duas coisas:

1. Compilar o projeto
2. Conseguir rodar a versão mais recente (*master*) lado a lado com a versão de produção

Importante:

1. Consulte a documentação de desenvolvimento!
2. Não precisamos ser especialistas para contribuir!

Exemplo 1: CMakeLists.txt

```
# CMakeLists files in this project can
# refer to the root source directory of the project as ${HELLO_SOURCE_DIR} and
# to the root binary directory of the project as ${HELLO_BINARY_DIR}.
cmake_minimum_required (VERSION 2.8.11)
project (HELLO)

# Recurse into the "Hello" and "Demo" subdirectories. This does not actually
# cause another cmake executable to run. The same process will walk through
# the project's entire directory structure.
add_subdirectory (Hello)
add_subdirectory (Demo)
```

Exemplo 1: CMakeLists.txt

```
# Create a library called "Hello" which includes the source file "hello.cxx".
# The extension is already found. Any number of sources could be listed here.
add_library (Hello hello.cxx)

# Make sure the compiler can find include files for our Hello library
# when other libraries or executables link to Hello
target_include_directories (Hello PUBLIC ${CMAKE_CURRENT_SOURCE_DIR})
```

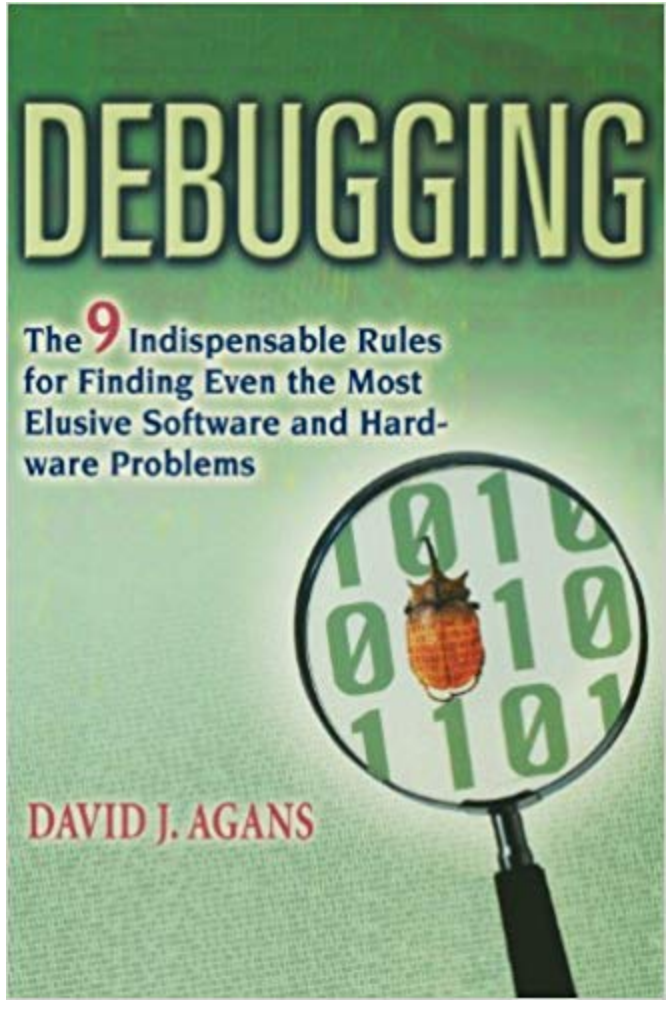
Exemplo 1: CMakeLists.txt

```
# Add executable called "helloDemo" that is built from the source files
# "demo.cxx" and "demo_b.cxx". The extensions are automatically found.
add_executable (helloDemo demo.cxx demo_b.cxx)

# Link the executable to the Hello library. Since the Hello library has
# public include directories we will use those link directories when building
# helloDemo
target_link_libraries (helloDemo LINK_PUBLIC Hello)
```

Exemplo 2: setuptools

```
from setuptools import setup, find_packages
setup(
    name="HelloWorld",
    version="0.1",
    packages=find_packages(),
    scripts=['say_hello.py'],

    # Project uses reStructuredText, so ensure that the docutils get
    # installed or upgraded on the target machine
    install_requires=['docutils>=0.3'],

    package_data={
        # If any package contains *.txt or *.rst files, include them:
        '': ['*.txt', '*.rst'],
        # And include any *.msg files found in the 'hello' package, too:
        'hello': ['*.msg'],
    },

    # metadata to display on PyPI
    author="Me",
    author_email="me@example.com",
    description="This is an Example Package",
    license="PSF",
    keywords="hello world example examples",
    url="http://example.com/HelloWorld/",   # project home page, if any
    project_urls={
        "Bug Tracker": "https://bugs.example.com/HelloWorld/",
        "Documentation": "https://docs.example.com/HelloWorld/",
        "Source Code": "https://code.example.com/HelloWorld/",
    }

    # could also include long_description, download_url, classifiers, etc.
)
```

Debugging


DEBUGGING
The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hard-ware Problems
DAVID J. AGANS

9 regras de debug

1. UNDERSTAND THE SYSTEM
2. MAKE IT FAIL
3. QUIT THINKING AND LOOK
4. DIVIDE AND CONQUER
5. CHANGE ONE THING AT A TIME
6. KEEP AN AUDIT TRAIL
7. CHECK THE P LUG
8. GET A FRESH VIEW
9. IF YOU DIDN'T FIX IT, IT AIN'T FIXED

Destrinchando as 4 regras

1. UNDERSTAND THE SYSTEM:
    - Compile o projeto
    - Rode sua versão
    - Saiba dizer onde estão os fontes e por qual critério estão organizados
2. MAKE IT FAIL
    - Reproduza o bug

Destrinchando as 4 regras

3. QUIT THINKING AND LOOK
    - Encontre no código onde o bug pode estar
    - Comece geral (em qual arquivo está a funcionalidade?) e vá restringindo (em qual função o bug "explode"?)
4. DIVIDE and CONQUER
    - Faça uma lista de tarefas contendo modificações que você acha que resolveriam o problema e que você precisa aprender para fazê-las
    - Ordene-as de acordo com sua facilidade.

Como se encontrar no projeto

Problema: Como encontro em qual arquivo mexer?

Ferramenta: comando `grep` permite buscar por strings em todos os arquivos de uma pasta.

Solução:

> `$ grep [OPTIONS] PATTERN FILES

- `PATTERN`: expressão regular
- `FILES`: lista de diretórios ou arquivos

Como se encontrar no projeto

Exemplo 1: buscar todos arquivos nas pasta atual (.) e subpastas com o texto "dialog" ignorando maiúsculas/minúsculas.

```
$ grep -r -i dialog .
```

Mais info:

- `man grep` (disponível online)
- Seção *Regular Expressions* é particularmente relevante

Como se encontrar no projeto

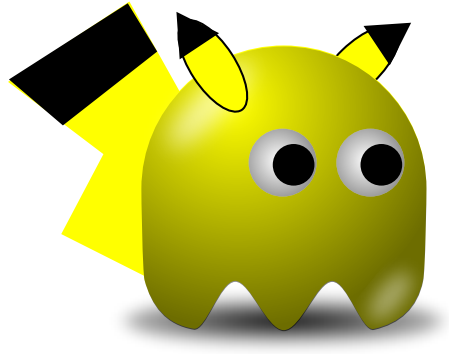
Exemplo 2: Listas todos os arquivos *.cpp* que fazem algum include

> `$ grep -r --include "*cpp" "#include" .

```
[igor@haute-normandie supercomp-2]$ grep -r --include "*.cpp"  "#include" .
./github/multi-core/03-intro-omp/calculo_pi.cpp:#include <stdio.h>
./github/multi-core/03-intro-omp/calculo_pi.cpp:#include <omp.h>
./github/multi-core/03-intro-omp/exemplo1.cpp:#include <iostream>
./github/multi-core/03-intro-omp/exemplo1.cpp:#include <omp.h>
./github/multi-core/02-fork-join/exemplo1-threads.cpp:#include <thread>
./github/multi-core/02-fork-join/exemplo1-threads.cpp:#include <iostream>
./github/simd/funcs.cpp:#include <math.h>
./01-introducao/src/teste.cpp:#include "funcs.cpp"
./01-introducao/src/teste.cpp:#include <iostream>
./01-introducao/src/sqrt_simd.cpp:#include <x86intrin.h> //Extensoes SSE
./01-introducao/src/sqrt_simd.cpp:#include <bits/stdc++.h> //Bibliotecas STD
./01-introducao/src/funcs.cpp:#include <math.h>
./01-introducao/src/funcs.cpp:#include <iostream>
./01-introducao/src/funcs.cpp:#include <chrono>
```

Próximos passos

1. Escolha em definitivo sua issue
2. Envie uma mensagem no Github/Issue tracker mostrando interesse
3. Registre o projeto escolhido com a Skill "Eu escolho você". No campo *proof*, envie a url da skill desejada.



Desenvolvimento Aberto

Debugging

Igor dos Santos Montagner (igorsm1@insper.edu.br)