Cling – The New C++ Interpreter for ROOT 6

V. Vassilev, P. Canal, A. Naumann, P. Russo
Role of C++ Interpreter in ROOT

- Load/Store C++ objects
- Runtime Dynamism
  - TFile::Open("http://...")
  - gDirectory->Get("hist")
  - python runReco.py
- Fast Prototyping
Role of C++ Interpreter in ROOT

ROOT uses the interpreter much more than one would expect:
Cling Is Better Than CINT

- Full C++ support
  - STL + templates
  - Path to C++11
- Correctness
- Better type information and representations
- Always compile in memory
- Much less code to maintain
Cling's Dual Personality

- An interpreter – looks like an interpreter and behaves like an interpreter
  
  *Cling follows the read-evaluate-print-loop (repl) concept.*

- More than interpreter – built on top of compiler libraries (Clang and LLVM)
  
  *Contains interpreter parts and compiler parts. More of an interactive compiler or an interactive compiler interface for clang.*

---

No need to compile Cling/ROOT with Clang/LLVM
Cling Uses Clang & LLVM

"The LLVM Project is a collection of modular and reusable compiler and toolchain technologies..."
Cling's Codebase

- Other ROOT – 1400K SLOC*
- CINT+Reflex – 230K SLOC*
- Cling – 7K SLOC*

* No testsuites included.
Credits: generated using David A. Wheeler's 'SLOCCount'

vvassilev / CHEP 2012.05.21
Cling's Codebase

Other ROOT – 1400K SLOC*
CINT+Reflex – 230K SLOC*
Cling – 7K SLOC*

externals:
LLVM + Clang – 800K SLOC*

* No testsuites included.
Credits: generated using David A. Wheeler's 'SLOCCount'
Challenges

- Incompatible concepts like compilation and interpretation
  
  *Many tasks that are trivial for an interpreter become a nightmare for a compiler.*

- Make C++ usable at the prompt
  
  *Incorporate the experience we have with CINT. First step: adopt the successful usability extensions from CINT.*
Cling in a Nutshell
Extending C++ Language

We want to be able to run statements

```cpp
void wrapper() {
    sin(12);
}
void wrapper1() {
    int i = 12;
}
void wrapper2() {
    sin(i);
}
```
Extending C++ Language

✦ Wrap the input
✦ Look for declarations
✦ Extract the declarations one level up, as global declarations

```c
int i = 12;

void wrapper1() {
  int i = 12;
  printf("%d\n", i);
}

void wrapper2() {
  printf("%f\n", sin(i));
}
```

```c
[cling]$ int i = 12; printf("%d\n", i);
[cling]$ printf("%f\n", sin(i));
```
Streaming Execution Results

[cling]$ \text{int} \ i = 12
\text{(int) 12}
[cling]$ \text{sin}(i)
\text{(double const) -5.365729e-01}

No semicolon (;)
Error Recovery

- Filled input-by-input
- Incorrect inputs must be discarded as a whole

```
*** CLING ***
* Type C++ code and press enter to run it *
* Type .q to exit *

[cling]$ int i; ERROR_HERE; int j;
input_line_4:2:9: error: use of undeclared identifier 'ERROR_HERE'
    int i; ERROR_HERE; int j;
    ^

[cling]$ i
input_line_5:2:2: error: use of undeclared identifier 'i'
i
    ^

[cling]$
```
Work in progress: Code Unloading

```cpp
// Calculator.h
class Calculator {
    int Add(int a, int b) {
        return a - b;
    }
    ...
};
```

```
// Calculator.h
class Calculator {
    int Add(int a, int b) {
        return a + b;
    }
    ...
};
```

* What's That Function
Late Binding

\{
TFile* F = 0;
if (is_day_of_month_even())
    F = TFile::Open("even.root");
else
    F = TFile::Open("odd.root");
\}

hist->Draw();

\+ Opens a dynamic scope. It tells the compiler that \texttt{cling} will take over the resolution of possible unknown identifiers.

\checkmark Defined in the root file

\x The root file is gone. Issue an error.
Late Binding

```cpp
{
    TFile* F = 0;
    if (is_day_of_month_even())
        F = TFile::Open("even.root");
    else
        F = TFile::Open("odd.root");
    gCling->EvaluateT<void>("hist->Draw()");
}

hist->Draw();
```

- Tell the compiler the identifier will be resolved at runtime
- Wrap it into valid C++ code
- Partially recompile at runtime
Challenges

- **Error recovery**
  
  Even though the user has typed wrong input at the prompt cling must survive, i.e. issue an error and continue to work.

- **Initialization of global variables**
  
  Cling depends on global variables, which need to be initialized. However, the global variables continue to be added (potentially) with every input line.

- **Late binding**
  
  Cling needs to provide a way for symbols unavailable at compile-time a second chance to be provided at runtime.

- **Value printer**
  
  The interactive mode obeys the repl concept and there should be way of easy print value and type of expression in a user-extensible way.

- **Running statements**
  
  CINT-specific C++ extension improving the user interaction with the interpreter from the terminal.
Cling In The World

- Announced in July 2011 as working C++ interpreter
- Cling and OpenGL
  - http://www.youtube.com/watch?v=eoluqLNvzFs
- Cling and QT
  - http://www.youtube.com/watch?v=BrjV1ZgYbbA
- MATLAB to C++ translator
- Regular bug reports from outside HEP
Cling In ROOT
Dictionaries

Describe compiled code

* Incarnation of the type information and reflection in ROOT. The only way of crossing the border between interpretation and compilation.

CINT and Reflex dictionaries:

* Double the size of the libraries
* Multiple copies of the dictionary data in the memory
* Incompatible reflection formats
* Do not cover 100% of C++
CINT Dictionary Use

Dictionary.(h|cxx)

- TClass
- ClassDef
- Call Stubs
- Reflection Info

Source of ROOT's class description

Member functions injected by the ClassDef macro

Target-independent, normalized signatures used to call compiled functions.

Teaches CINT what a class contains

Dictionary Size (SLOC)
- 23%
- 6%
- 10%
- 61%
LLVM Just-in-time Compiler

- Compiles lazily just before the function is called
- Has full knowledge about the target architecture
- Much faster than interpreting the LLVM Bit Code

LLVM Bit Code → Cling → Clang → LLVM JIT → Machine Code (x86, Alpha, ...) → Compiled libs(.so)
ROOT 6 in November
See plenary by Fons on Wednesday

Windows support to arrive after 6.00
Work in progress in clang

“genreflex”-compatible dictionary generator after 6.00
Reduce size of dictionaries

- Mid term: Call Stubs & Reflection Info goes away!
- Long term: No dictionaries at all

Compiled TFormula
Additional Features

- World class performance and optimizations
- OpenCL
- C
- Objective C[++]
- ...

In action and the truth

Come and see at our booth
Fons's plenary on Wednesday

Google Tech Talk

LLVM Euro Dev Dev Meeting 2011


LLVM Dev Meeting 2011


Thank you!