cling

Axel Naumann (CERN), Philippe Canal (Fermilab), Paul Russo (Fermilab), Vassil Vassilev (CERN)
Creating cling, an interactive interpreter interface for clang
cling?

- C++* interpreter*
  *: not really

- interactive, i.e. prompt

- used like python, bash and php, but C++

- cling: “C++ LLVM-based Interpreterg”
context:

- CERN’s LHC: petabytes of serialized C++ objects / year
- analyzed by >10,000 physicists world-wide
- performance counts
- approx 20M LOC C++
legacy:

- experience through CINT: C++ interpreter, 15 years old
- main use in data analysis framework @ http://root.cern.ch
- limitations, limitations... (parsing, design)
- re-write with clang the obvious solution!
C++ dialect:

- statement at translation unit scope
- implicit #includes
- automatic loading of dynamic libraries
- implicit auto keyword
- dynamic scopes

```cpp
void f()
{
  Klass o;
  o.f();
}

void f()
{
  File f("f.root");
  hist->Draw();
}
```

i = 12;
“interpreter”?

not lli, not a traditional interpreter

instead:

runtime evaluation

pseudo-instantaneous response
(c.f. compile + link + load + disk I/O)

ahead of time compilation where possible
big picture:

"C++" code with cling extensions

AST/
Sema
clang

ExeEng

Thursday, November 4, 2010
"C++" code with cling extensions

AST/Sema
clang

cling

ExeEng

big picture:
big picture:

“C++” code with cling extensions

AST/Sema

clang

cling

ExeEng
use cases!
use case: development

explorative development of algorithms:

$ .x myCode.C(47, "argument")$

// myCode.C:
void myCode(int, const char*){}

edit, run, unhappy about output:

void f() {}

in CINT, re-interpret

300x faster than full recompile, link, load!
use case: dev't (2)

- code looks the same wherever it is (C++!)
- easy migration from physicist to framework
- simple transition interpreted / compiled:

  $ .x myCode.C
  $ .x myCode.C+
use case: signal / slot

- intuitive function call:
  ```c++
  fCheck->Connect("Toggled(Bool_t)", "MyClass", this,"CallMe(12)");
  ```
- runtime binding
- runtime parameter values
- easily extensible (a string!)
use case: plugins

- function call through string

```c
void P110_THDFSFile() {
    gPluginMgr->AddHandler("TFile", "^hdfs:", "THDFSFile",
        "THDFSFile(const char*,Option_t*,const char*,Int_t)");
}
```

- can depend on runtime

- loose, optional coupling of libraries
use case: python binding

- C++ reflection + interpreter: control over objects and calls
- runtime discovery of types + functions: no stubs
- both ways: python <-> C++
use case: python binding

// Create a one dimensional function and draw it
fun1 = new TF1("fun1", "abs(sin(x)/x)", 0, 10);
fun1->Draw();

from ROOT import TF1
# Create a one dimensional function and draw it
fun1 = TF1('fun1', 'abs(sin(x)/x)', 0, 10)
fun1.Draw()

use the same (C++) library
and more use cases:

- configuration management
- reflection for serialization, documentation
- beyond high energy physics:
  - AI for computer game
  - remote configuration of integrated devices
- add your own (e.g. debugger?)
dissecting clinging!
parts:

- interpreter: parses, JITs, executes (C++)
- meta processor: e.g. unload source file, debug (steer)
- user interface: e.g. prompt loop, exception handling
prompt $$

think of

$
prompt $ $ 

think of

$ int i(17) $

$ $
prompt $ 

think of

$ int i(17)
$ i=42
$
$ int i(17)
$ i=42
$ int f() {
  ...

prompt $ 

think of 

$ \text{int } i(17) \\
$ \text{int } i=42 \\
$ \text{int } f() \{ \\
\ldots \text{ return} \\
\ldots $
$\textit{prompt}$ $\$

```
$\texttt{int i} (17) \\
$\texttt{i=42} \\
$\texttt{int f() \{} \\
    \texttt{... return} \\
    \texttt{... ++i;} \\
$\texttt{f();}$
```

think of
prompt $ $

1. transform input: declarations vs. statements
2. add to existing AST (“ever-growing AST”)
3. remap globals
4. run only new initializers
5. call statement stub

```cpp
using namespace __cling;
namespace __cling {
    int i={17};
    int f() { return ++i; }
}
void clinging_prompt_0() {i=42;}
void clinging_prompt_1() {f();}
```
library auto-loading:

1. intercept on unresolved symbol
2. look up in our symbol ➔ library map
3. dlopen
4. rewire global mapping to symbol
#includes optional:

- PCH for everything
- multiple PCHs
- need to prune PCH overlaps / vetoed types via AST manipulation and dependency analysis
compiled <-> interpreted

- recursive* ExecutionEngine / JIT invocation

```cpp
void f() {
  cling::Interpret("f();");
}
```

```
$ .x f.C
```

* yes, perfect recursion!

- symbol resolution into libraries
dynamic scoping:

- file opens a dynamic scope
- unknown id: query file's objects
- but need valid AST:
  - mark unknown identifiers dependent
- transform into delayed evaluation:

```cpp
void f()
{
  File f("f.root");
  hist->Draw();
}
```
dynamic scoping:

- file opens a dynamic scope
- unknown id: query file's objects
- but need valid AST:
  - mark unknown identifiers dependent
- transform into delayed evaluation:

```cpp
void f()
{
  File f("f.root");
  hist->Draw();
}
```

```cpp
void f()
{
  File f("f.root");
  cling::eval("hist->Draw();");
}
```
un-/reloading

// Struct.h, v1.0
struct S {
    int f() { return 17; }
};

// Struct.h, v2.0
struct S {
    int f() { return 42; }
};

$ .L Struct.h
$ S o
$ o.f()
17
$ .U Struct.h

// edit Struct.h

$ .L Struct.h
$ S o
$ o.f()
42
un-/reloading

- checkpoint ever-growing AST (undo points)
- prune top level decls if dependency analysis allows
- global mapping magic
conclusion
cling:

- ahead-of-time compiler
- extends C++ for ease of use as interpreter
- interfaces clang
- isn’t finished yet!
main ingredients:

- interactive prompt
- library auto-loading
- #includes optional
- dynamic scoping
- calling compiled <-> interpreted
- un- / reloading of libraries and source
wish list:

- growing MemoryBuffer with “top level decl not finished” callback
- AST dependency analysis:
  - AST still valid after removal of node N?
- shortcut JIT -> shared lib
santa:

- can anybody help us with AST dependency graph?
  need to do it ourselves?

- possible use cases: code dependency / dead code analysis, refactoring
status:

- some things work: simple prompt, library <-> interpreter (even recursion), auto-loading libs
- currently attacking dynamic scope, extending prompt, interface to high energy physics data analysis software
- for the rest, you have seen the plan!
where we are:

- \texttt{svn co http://root.cern.ch/svn/root/branches/dev/cling}

- code’s copyright / license points to clang’s

- would like to move into clang repo once cling is a bit more mature / usable – hopefully within this calendar year (santa!)