Implement value printing in clang-repl

Jun Zhang
jun@junz.org

Mentors:
Vassil Vassilev & David Lange
What are the goals of the project?
Support better pretty printing in clang-repl

```
clang-repl> int x = 42;
clang-repl> x
(int&) 42

clang-repl> "Hello, interactive C++!"
(const char [24]) "Hello, interactive C++!"

clang-repl> std::vector<int> v {1,2,3};
clang-repl> v
(std::vector<int>) {1,2,3}
```
Better integration between compile/interpreted code

https://github.com/root-project/cling/blob/master/tools/demo/cling-demo.cpp#L20-L45
Implementation

1. Determine if this is an expression need to be print.
2. Synthesize a value.
3. Print value using Value::dump()
4. fall back to a runtime call if all fails.

clang-repl> int x = 42;
clang-repl> x
(int&) 42
Current Status of clang-repl

After https://reviews.llvm.org/D127284 Clang now has an extension that supports statements on global scope in incremental mode.

```
clang-repl> #include <stdio.h> // for printf
clang-repl> int x = 42;
clang-repl> printf("x = %d\n",x); // Previously will fail to compile.
    // We have to do `auto r = printf(..);` as a workaround.
x = 42 // Now it works!
```
Capture the expression result

Since the IncrementalParser is powered by Clang parser, we need to teach Clang parser to recognize the pattern.

```cpp
// Otherwise, eat the semicolon.
ExpectAndConsumeSemi(diag::err_expected_semi_after_expr);
return handleExprStmt(Expr, StmtCtx);
```
Synthesize the value

We need to create a Value object to carry the result of the expression. Thus, we manually inject some code:

```c
void SetValue(void* OpaqueType, T Expr, Value* OutValue);
```
Print it!

1. Primitive types
   Val->dump();

2. “Complex” types like STL component or user-defined struct/class
   Inject code again to a function like PrintValueRuntime, which lives in a header that is processed by the JIT ahead of time.
   
   clang-repl> #include “PrintValue.h”
   
   clang-repl> PrintValueRuntime(x);
Road map

1. Write a detailed RFC and post it in the LLVM Discourse Group (Ongoing)
2. Implement the Value class and its dump method
3. Support value printing for builtin types
4. Support value printing for non-primitive types
5. Support value printing for temporaries
Q & A?
Thanks!