30 Mar `22

IRIS-HEP Compiler Research

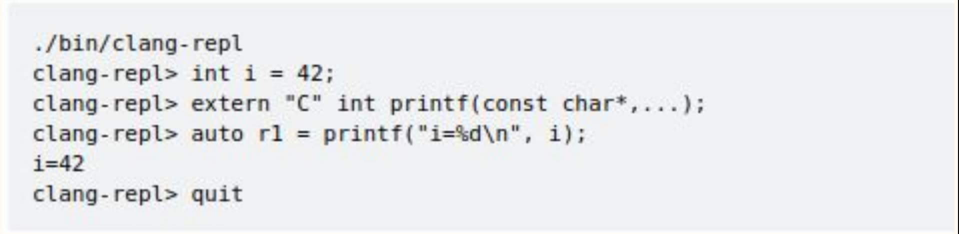
Enhance the incremental compilation error recovery in clang and clang-repl

Mentors: Dr. Vassil Vassilev, Dr. David Lange
Student: Purva Chaudhari

iris hep
Institute for Research & Innovation in Software for High Energy Physics

Content

Clang-Repl Overview



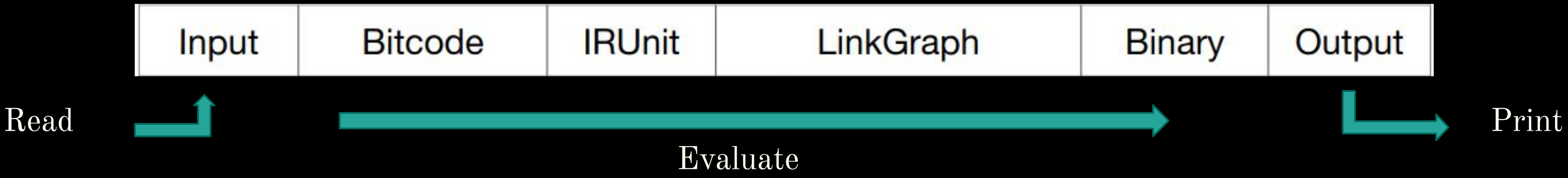
```
./bin/clang-repl
clang-repl> int i = 42;
clang-repl> extern "C" int printf(const char*,...);
clang-repl> auto r1 = printf("i=%d\n", i);
i=42
clang-repl> quit
```

Ref: LLVM review D96033

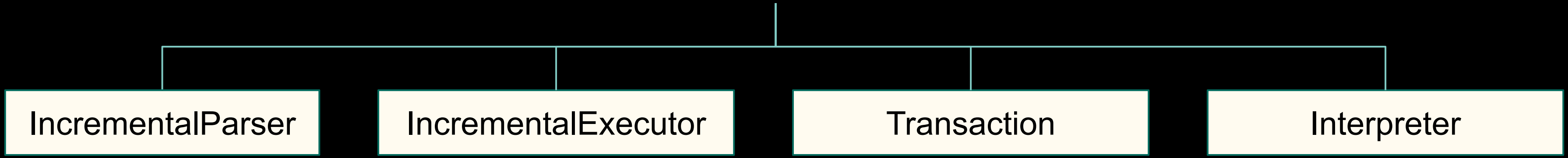
- Cling built on top of LLVM and clang was initially developed to enable interactive high-energy physics analysis in a C++ environment.
- Clang-Repl is a new tool which incorporates Cling in the Clang mainline

How Clang-Repl works

Pipeline



Code Infrastructure



Error Recovery in Clang-Repl

- Translation unit in clang can be split into a sequence of partial translation units (PTUs)
- Owning PTU is not always the most recent PTU and processing a PTU might extend an earlier PTU.
- Clang-repl recovers from errors by disconnecting the most recent PTU and update the primary PTU lookup tables

```
clang-repl> int i = 12; error;
In file included from <<< inputs >>>:1:
input_line_0:1:13: error: C++ requires a type specifier for all
declarations
int i = 12; error;
           ^
error: Parsing failed.
```

Ref: Vassil V. Commit - Implement partial translation units and error recovery.

Progress till now

- Added recovery flag
- Reuse Clang Tests
- Fixing bugs

1. Recovery Flag

- The recovery mode would enable reusing some of the clang tests to clang-repl behaviour tests.
- Running a behaviour test in recovery mode stores the current PTU, processes the file and restores back to the stored current PTU
- The recovery flag is based on the error recovery logic of the clang-repl

```
// RUN: clang-repl -recovery -Xcc -fsyntax-only -Xcc -verify
%S/../Sema/address-constant.c
// RUN: clang-repl -recovery -Xcc -fsyntax-only -Xcc -verify
%S/../Sema/arg-scope.c
//expected-no-diagnostics
```

2. Testing

- ➢ Currently some simple clang tests have been re-used and included in clang-repl
- ➢ The tests are llvm lit //expected-no-diagnostics
- ➢ Mostly tests with -fsyntax -verify have been included with a few additional support for other flags and std c++ versions
- ➢ The tests do not yet support the -triple flag

Tests Included	**Passing** (+ → tests fail in parsing but pass in interactive mode)	**Failed**
Sema	23+1	7
SemaCXX	118+2	1
	141+3	**8**

3. Bug fix for error recovery

Resolved recovery for variable to be reused in case of error occurred in the same line of parsing (when it was a subsequent parsing).

```
clang-repl> int j=9; err;
input_line_2:1:10: error: C++ requires a type
specifier for all declarations
int j=9; err;
      ^
error: Parsing failed.
clang-repl> int j = 9;
input_line_3:1:5: error: redefinition of 'j'
int j = 9;
```

Before

```
clang-repl> int j=3; err;
In file included from <<< inputs >>>:1:
input_line_1:1:10: error: C++ requires a type
specifier for all declarations
int j=3; err;

clang-repl> int j=3;
clang-repl> ^C
```

After

Further goals

- Handle failing tests
- Add folders to reuse clang tests of CodeGen, CodeGenCXX, Lexer, Parser
- Checking cases for template-instantiation

Failing tests

1. Redefinition error for __typeof__ cases

```
struct {unsigned x : 2;} x;
__typeof__((x.x+=1)+1) y;
__typeof__(x.x<<1) y;
int y;
```

2. Initialize a variable of with an rvalue of type 'void *'

```
char *a = (void*)(uintptr_t)(void*)&a;
```

3. Enums handling

```
enum A { A1, A2, A3 };
typedef enum A A;
void test() {
  A a;
  a++;
  a--;
  ++a;
  --a;
  a = a + 1;
  a = a - 1;
}
```

Failing tests

4. Use of `this` keyword

```
typedef struct {
    char *str;
    char *str2;
} Class;

typedef union {
    Class *object;
} Instance
__attribute__((transparent_union));

__attribute__((overloadable)) void
Class_Init(Instance this, char *str,
void *str2) {
    this.object->str  = str;
    this.object->str2 = str2;
}
```

5. Redeclaration of static int

```
static int a;
int bar() {
  extern int a;
  return a;
}
static int a;
```

Thank You