Compiler Construction

Write a GCC plugin

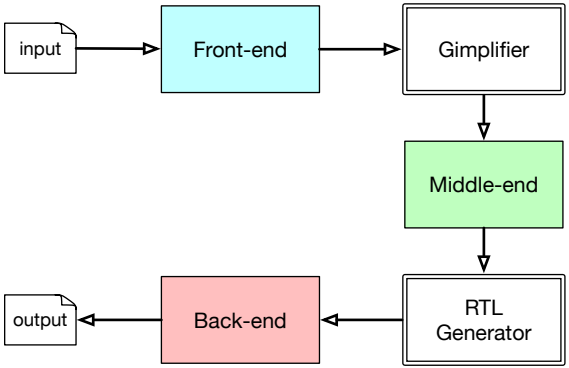
Goal

How to write a GCC plugin

Objectives:

- Add your own annotations
- Extract some relevant information
- Write an additional optimization pass
- Write DSL
- Generate your own warning
- Writing coverage/documentation tools
- ...

GCC overview

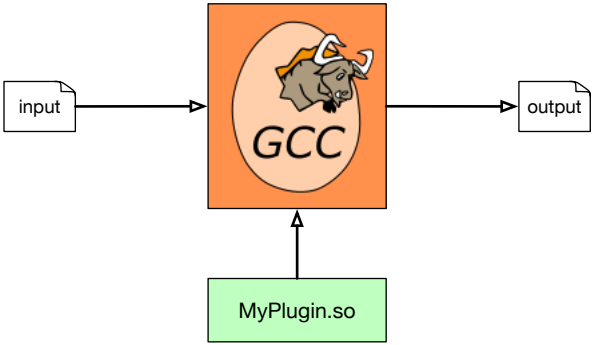


GCC is big !

Around than 2,000,000 LOC

More than 200 transformation passes
⇒**Hopefully there is a pass manager!**

GCC plugin



Which plugin?

```c
#include <stdio.h>

int dummy(int n)
{
  return 1 + 5;
}

int main()
{
  printf("%d\n", dummy(1));
  return 0;
}
```

```c
#include <stdio.h>

int dummy(int n)
{
  return 1 + 10/*<- HERE*/;
}

int main()
{
  printf("%d\n", dummy(1));
  return 0;
}
```

Which plugin?

```c
#include <stdio.h>

int dummy(int n)
{
  return 1 + 5;
}

int main()
{
  printf("%d\n", dummy(1));
  return 0;
}
```

```c
#include <stdio.h>

int dummy(int n)
{
  return 1 + 10/*<- HERE*/;
}

int main()
{
  printf("%d\n", dummy(1));
  return 0;
}
```

Which pass?

- PLUGIN_START_PARSE_FUNCTION
- PLUGIN_FINISH_PARSE_FUNCTION
- PLUGIN_PASS_MANAGER_SETUP
- PLUGIN_FINISH_TYPE
- PLUGIN_ATTRIBUTES
- PLUGIN_START_UNIT
- PLUGIN_PRAGMAS
- PLUGIN_ALL_PASSES_START
  PLUGIN_ALL_PASSES_END
- PLUGIN_FINISH_DECL
- PLUGIN_FINISH_UNIT
- PLUGIN_PRE_GENERICIZE
- PLUGIN_FINISH
- PLUGIN_INFO
- PLUGIN_GGC_START
- PLUGIN_GGC_MARKING
- PLUGIN_GGC_END
- PLUGIN_REGISTER_GGC_ROOTS
- PLUGIN_ALL_IPA_PASSES_START
  PLUGIN_ALL_IPA_PASSES_END
- PLUGIN_OVERRIDE_GATE
- PLUGIN_PASS_EXECUTION
- PLUGIN_EARLY_GIMPLE_PASSES_START
  PLUGIN_EARLY_GIMPLE_PASSES_END
- PLUGIN_NEW_PASS
- PLUGIN_INCLUDE_FILE

https://gcc.gnu.org/onlinedocs/gcc-9.1.0/gccint/Plugin-API.html

Writing a plugin (1/5): preliminaries

```
#include <iostream>
// This is the first gcc header to be included

#include "gcc-plugin.h"
#include "plugin-version.h"
#include <tree.h>
#include <print-tree.h>

static struct plugin_info my_gcc_plugin_info =
       { "1.0", "This is a very simple plugin" };

#define PLUGIN_NAME "CMP1-plugin"

int plugin_is_GPL_compatible;
```

Writing a plugin (2/5): entry point

```
int plugin_init (struct plugin_name_args *plugin_info,
                 struct plugin_gcc_version *version)
{
  if (!plugin_default_version_check (version, &gcc_version))
    {
      std::cerr << "This GCC plugin is for version "
                << GCCPLUGIN_VERSION_MAJOR
                << "." << GCCPLUGIN_VERSION_MINOR << "\n";
      return 1;
    }

  register_callback(plugin_info->base_name,
                    /* event */ PLUGIN_INFO,
                    /* callback */ NULL,
                    /* user_data */ &my_gcc_plugin_info);
```

Writing a plugin (3/5): arguments and callback

```
  std::cerr << "Number of arguments of this plugin:"
            << plugin_info->argc << "\n";

  // Walk Parameters
  for (int i = 0; i < plugin_info->argc; i++)
    {
      std::cerr << "Argument " << i
                << ": Key: " << plugin_info->argv[i].key
                << ". Value: " << plugin_info->argv[i].value << "\n
    }

    // Register Callback
    register_callback(plugin_info->base_name,
            PLUGIN_PRE_GENERICIZE,
            pre_genericize_callback, /* user_data */ NULL);
  return 0;
}
```

Writing a plugin (4/5): look for the function

```
static void pre_genericize_callback(void *gcc_data,
                                    void *user_data){
    printf(":pre_generic: "PLUGIN_NAME"\n");

    tree fndecl = (tree) gcc_data;
    if (TREE_CODE(fndecl) == FUNCTION_DECL) {
        tree id = DECL_NAME(fndecl);
        const char *fnname = id ? IDENTIFIER_POINTER(id)
                               : "<unnamed>";

        if (strcmp(fnname, "dummy"))
            return ;
```

Writing a plugin (5/5): perform changes

```
        tree fnbody = DECL_SAVED_TREE(fndecl);

        if (TREE_CODE(fnbody) == RETURN_EXPR) {
            tree t = TREE_OPERAND(fnbody, 0);
            t = TREE_OPERAND(t, 1);
            // debug_tree(t); //<- BEFORE CHANGES
            TREE_OPERAND(t, 1) =
                build_int_cst(integer_type_node, 10);
            // debug_tree(t); //<- AFTER CHANGES
        }
    }
}
```

Test a pluging (1/2)

```
# FIXME Your GCCDIR
GCCDIR = /home/etienne/gcc-plugin/gcc-install/bin

CXX = $(GCCDIR)/g++
# Flags for the C++ compiler: -fno-rtti is required for GCC plugins
CXXFLAGS = -std=c++17 -Wall -fno-rtti

# Determine the plugin-dir and add it to the flags
PLUGINDIR=$(shell $(CXX) -print-file-name=plugin)
CXXFLAGS += -I$(PLUGINDIR)/include

# top level goal: build our plugin as a shared library
all: cmp1.so

cmp1.so: cmp1.o
        $(CXX) $(LDFLAGS) -shared -o $@ $<
```

Test a pluging (2/2)

```
cmp1.o : cmp1.cc
	$(CXX) $(CXXFLAGS) -fPIC -c -o $@ $<

clean:
	rm -f cmp1.o cmp1.so

check: cmp1.so
	$(CXX) ./test.cc -o a.out && ./a.out
	$(CXX) -fplugin=./cmp1.so -fplugin-arg-cmp1-arg1=CMP1 \
	                          -fplugin-arg-cmp1-arg2=ROCKS \
	       ./test.cc -o my_a.out && ./my_a.out


.PHONY: all clean check
```

Writing a plugin (5/5): perform changes

```
6
Number of arguments of this plugin:2
Argument 0: Key: arg1. Value: CMP1
Argument 1: Key: arg2. Value: ROCKS
:pre_generic: CMP1-plugin
:pre_generic: CMP1-plugin
11
```

Summary