CS 502 – Fall 2015

Project 1 AST Traverser in GCC

(posted Oct. 9, 2015)

Deadline: Nov. 1, 2015, 11:59pm

1. Project Description

1.1 Overview

This project requires the students to implement a **traverser** in GCC-4.7.0 compiler. This traverser navigates through the abstract syntax tree (AST) constructed for the C program being compiled and, based on which, collects statistics about the code.

AST is a high level intermediate representation. This project will enhance students' understanding of the implementation of a programming language, by knowing how a program can be represented in a compiler. By implementing the traverser, we expect the students to get familiar with GCC's AST representation that is named GENERIC in the GCC community. Knowing how statements are represented in GENERIC, we will be able to implement flow analysis in Project 2 in this course.

In order to reduce the mundane aspects and the overall load of the project, only a subset of C is required (see Appendix). Students are welcome to implement a somewhat bigger subset as they like. However, they must not simply copy from existing traversers from any source. (Some of the open source GCC versions have traversers for a much bigger subset or the full C language. The students are welcome to learn from those examples on their own, but *copying is prohibited*.)

The project is to be done individually, using any of the XINU machines in the CS department.

1.2 GCC Overview

GCC is an integrated distribution of open-source compilers for several major programming languages, including C, C++, Java, and so on. For each language, there is a compiler, which is responsible for translating the source file into possibly optimized target machine code. For example, C compiler is “cc1” and Java compiler is “jc1”. CS502 projects will focus on the C compiler (i.e. cc1).

Illustrated in Fig 1, the architecture of GCC is divided into the following three main parts:

- **Front-End**: source language-dependent.
- **Middle-End**: both language-independent and target-independent.
- **Back-End**: target-dependent.

The project will be done at the end of the Front-End stage.

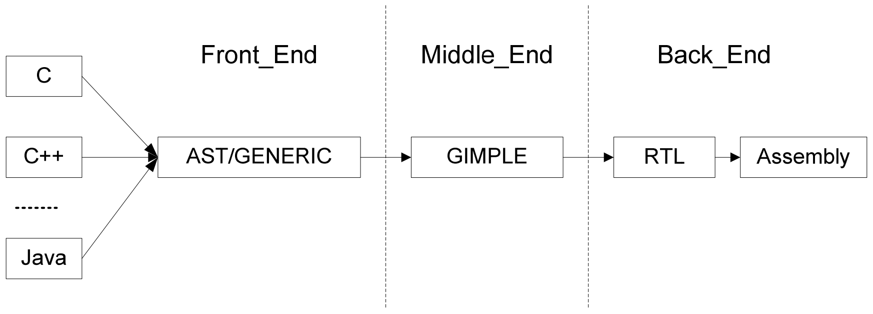

Fig 1. GCC internal framework

1.3 Project Call Chain in GCC-4.7.0

Fig 2 shows the main function call chain in GCC-4.7.0 source code related to Project1. The function call “cs502_proj1()”, which should be implemented in Project1, starts the traverser procedure.

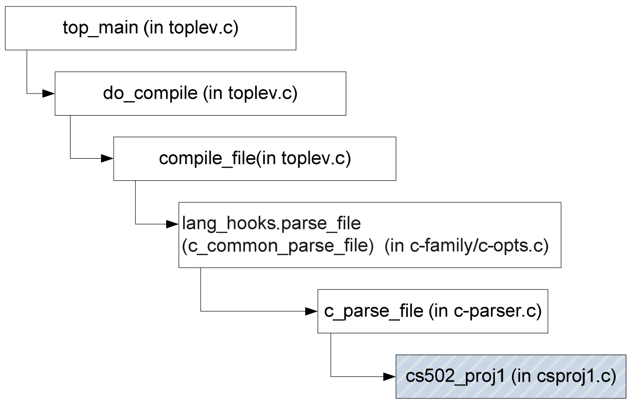
top_main (in toplev.c)
do_compile (in toplev.c)
compile_file(in toplev.c)
lang_hooks.parse_file (c_common_parse_file) (in c-family/c-opts.c)
c_parse_file (in c-parser.c)
cs502_proj1 (in csproj1.c)

Fig 2. Where project1 is invoked

1.4 Tree in GCC4.7.0

It is crucial for students to get familiar with the data structure of the AST (known as GENERIC in GCC's terminology) in gcc-4.7.0. The central data structure is **tree (**defined in **tree.h, tree.def)**. A **tree** is a pointer type, but the object to which it points may be of a variety of types. (GCC implements it with complicated union and struct definitions.) For each tree node, it has a TREE_CODE to indicate what kind of tree node it is. For example, a function is represented by a tree fn with

TREE_CODE(fn) = FUNCTION_DECL

And the high-level organization of fn (the function) approximately looks like what Fig 3 shows, where each node represents a tree node. More details of tree nodes please refer to the resources listed in Part 5 of this document.

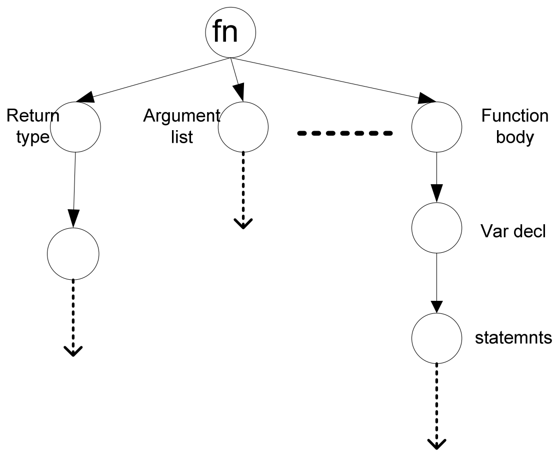

Fig 3. A possible high level organization of a tree with FUNCTION_DECL code

1.5 Other Suggestions

1) How to traverse all functions?

After the c parser, a call graph is build, represented by a global variable:

struct cgraph_node *cgraph_nodes

Each node in the call graph represent a function (Please refer to source code: cgraph.h). The following sample code snippet shows how to traverse the call graph and get every function:

```
struct cgraph_node *node;
 tree fn;
 for (node = cgraph_nodes; node; node = node->next) {
    fn = node->decl;//get a function
    ****
 }
```

2) How to handle global variable declaration?

In GCC-4.7.0, all global variables are stored in a global variable

struct varpool_node *varpool_nodes

The definition can be found in cgraph.h, varpool.c. The following code snippet shows how to traverse all global declarations:

```
Struct varpool_node *node;
For (node = varpool_nodes; node; node = node->next) {
        Tree decl = noe->decl; //get the global declaration
}
```

3) How to traverse and output a tree node (for debugging)?

This is exactly what students must investigate by doing this project However, one can shorten the learning time by taking a look at the function **dump_generic_node** in **tree-pretty-print.c** file. ***DO NOT** copy any code from those files!* You must understand how to traverse the tree and write your own code. Understanding the TREE structure and how to traverse it is the goal of Project1, and such understanding is also crucial for being able to complete Project 2.

2. How to Start

To start the project, students need to follow the steps listed below:

1. *Working under your scratch directory*. Otherwise, you may have “space is not enough” error during compiling. Please follow the following way:

   1) Login to any xinu machine, and now you are in your home directory.

   2) Run the following command: xinu 01% cd ~/scratch

   Then you will get into your scratch directory

   3) Please do your project under this scratch directory. This will be the same as working under your home directory. However, the files under your scratch directory are not backed up. Therefore, you are advised to keep an updated copy of important SOURCE code you have written in your home directory.

   4) Once you finish your project, copy all necessary files: Makefile, source codes, readme, project report, and any other files mentioned in our instruction, to your home directory (**No executable files, no softlink, and no intermediate files generated by compiling are allowed in your home directory. Failure to follow this instruction will result in penalty to your grade, because such a failure may overload the system and affect all**

other users.) Next, run "make clean" to further make sure all unnecessary files are removed.

5) Submit your project from your home directory, details of which is in Section 3 (titled “How to Submit”).

2. To set up the environment of Project1, copy and unzip the file “cs502_fall15_proj1.tar.gz" from the directory “/u/data/u3/cs502/Fall15”. A directory “cs502_fall15_proj1" will be created in your scratch directory, which will contain a soft link “gcc-4.7.0_src” and a directory “install”. The soft link “gcc-4.7.0_src” points to the gcc-4.7.0 source code directory on xinu, which is read-only. The “install” directory is your working directory.
3. Go to the subdirectory named “cs502_fall15/install/". (The entire set of GCC 4.7.0 files is very big. Therefore we share as many files as possible in order to reduce the storage requirement.) In this directory, students should write a source file named “csproj1.c" to implement the traverser. A template with an empty function cs502_proj1 and a sample Makefile is already provided in this directory. (Section 1.3 describes how to invoke cs502_proj1 in the your revised gcc). There is another template file named “csproj2.c”, which will be used for our second project. Please do **NOT** remove or modify it this time (Removing csproj2.c will cause your “make” to fail). After the traverser is written, run the command “make" to create your own version of program “cc1" which has the traverser features built in.

   Note: Other than “csproj1.c” and any other source files you create that contain new functions called from “csproj1.c”, the project should **NOT** modify any existing source files in GCC4.7.0 that is provided in the environment described above. As explained in Section 1.3, the “csproj1.c" file must contain a function “void cs502_ proj1()" from which the tree traversal is launched. If you create new source files in addition to “csproj1.c”, then you must add these file names to your Makefile, but you are not allowed to make any other changes to the Makefile .
4. After a successful make, run “.\cc1 some-file-name.c", where some-file-name.c is any C program that you want to use for testing, so that cc1 reads some-file-name-.c, constructs the AST, the symbol table and the call graph, and then, calls your implemented traverser to generate the statistics as defined in Section 4 below.

5. A sample test case, test1.c, is provided in the “/cs502_fall15/install” directory. A corresponding output file, test1.out, is also provided, against which you can compare your output.

3. How to submit

After the preparation made as specified in Section 2.1.4, run the following command to submit your files,

turnin –c cs502 –p proj1 [your project1 directory]

Your submitted directory must include:

1) All new source files.

2) The new Makefile.

3) A README file explaining anything you want the TA to know in order to re-produce your directory and generate your own cc1.

4) A project report in plain text. (The contents of this report are explained in the grading criteria in Section 4.

Please make sure to run “**make clean**” and make sure “**cc1**” is removed before submitting your working directory. Otherwise, your file will be too large to be accepted.

4. Output Format & Grading Criteria

Your AST traverser must collect the following statistics and write the result to an output file named “output.txt”. The output must be formatted strictly as described below. Otherwise you will lose points.

```
#functions: [count]
#global vars: [count]
#local vars: [count]
#statements: [count]
```

In the above, #local vars must include all local variables from all functions. Local variables declared in different functions are counted as different ones even if they may have the same name. (It is your responsibility to find a way to distinguish them after reading GCC documents). Similarly, #statements must include all statements from all functions. More specifically, if a statement has multiple branches, then the statement itself and the branch keywords (such as else, case, etc) will be counted as a single statement, but statements within each block associated with a branch will be counted as other statements. This continues with those blocks nested deeper. If anything remains unclear to the students, question should be posted on Piazza. A testing example will be posted on Piazza. Meanwhile, students are encouraged to share their own testing cases on Piazza.

Grading Criteria

Correctness: (80%)

The statistics generated by the traverser should be correct and we will evaluate the outputs with automatic scripts. Make sure your program generates output in the exact format specified above. Students will be given an opportunity to explain the discrepancy between their results and the expected ones, and grade adjustment may be made depending on the cause of the discrepancy.

Following the output Format: (5%)

Clean Submission: (5%)

Documentation: (10%)

Proper comments must be inserted in the new source files to make the code reasonably easy to understand.

In addition, each student must submit a project report that

a. describes which parts of the project (referring to the language subset) are completed and which parts are not

b. lists all the files that have created and explain those new functions and major new data structures (if any).

c. For each tree node that is analyzed, add a brief comment in your program. For example, for a tree node with code “FUNCTION_DECL”, a comment to add could be “This node represents a function”. The helps demonstrate that the student understands each node that is analyzed.

5. Resources

There are resources useful for Project1, especially the second one on the following list.

1) GCC website

http://gcc.gnu.org/

2) GCC 4.7 internal manual, Chapter11 GENERIC:

https://gcc.gnu.org/onlinedocs/gcc-4.7.0/gccint/

3) GCC 4.7 Source code: linked from "gcc-4.7.0_src".

To help students who may find these document difficult to navigate, we will post a few mini-tutorials as a starter.

6. Appendix: The Subset of ANSI-C for Project 1

(This description is subject to minor changes to clarify any doubts raised by students)

A.1 Block Structures

In this subset, a C program has a main function and zero or more other functions, all of which are placed in a **single** source file. The organization of a C program looks like this:

```
< global declarations of types and variables>
<type> function-1 (<parameter list>)
{
        <block body>
}
…..
<type> function-n (<parameter list>)
{
        <block body>
}
```

There are **NO** header files included (such as "#include<stdio.h>"), **No** global declarations between any two functions.

A.2 Statements

<block body> consists of local declarations and a list of statements. We consider the following kind of statements:

- Assignment statement,
- **FUNCTION** call,
- **RETURN** statement,
- **IF** statement (which may or may not contain an ELSE branch),
- **SWITCH** statement,
- **WHILE** statement, and

- **BLOCK**.

A BLOCK looks like:

```
{
        <block body>
}
```

Note: a WHILE statement will be represented by **IF** tree nodes and **GOTO** tree nodes in GCC's AST. This does not have an impact on the analysis outcome.

A.3 Expression

For expressions in Milestone 1, we have the following operators:

= || && ! == != < > <= >= + - * /

The atomic operands are **ID**s, **integers**, **real numbers** and **characters**.

A.4 Data types and declarations

Scalar types: **integer**, **float** and **character**.

Variables may be initialized in their declaration statements.

Appendix B: What are not in the subset for Project 1

In general, what is not explicitly stated in Appendix A will not be required.

To clarify potential doubts, based on feedbacks from previous semesters, we explicitly state that the following will not be covered in the subset.

1. Unions
2. Pointer arithmetic
3. Typedef statements
4. Type coercion, such as " f = (float) a;"
5. Included header files, such as "#include <stdio.h>"

NOTE: The following may be covered in Project 2 but not yet in Project 1

1. Arrays and structures
2. Pointer dereferences such as *p and p→ q