Compiler Assignment 1

Pritam Kumar Nath, 16615

Department of CSA, Indian Institute of Science, Bangalore

1 ABSTRACT

The main objective of the assignment was to design a pass for the clang compiler that would do a source to source transformation to add nested functions in native C language.

2 KEYWORDS

clang;make;ninja;ast;rewriter

3 TOOLS USED

1. clang 14(installed clang tools and clang tools extra)
2. make
3. Cmake
4. ninja

4 INTRODUCTION

4.1 *libtooling:*

LibTooling is a library to support writing standalone tools based on Clang.The pass was designed using libtooling.It runs FrontendAction on the code.

4.2 *RecursiveASTVisitor:*

RecursiveASTVisitor extracts the relevant information from the AST. The RecursiveASTVisitor provides hooks of the form $boolVisitNodeType(NodeType*)$ for most AST nodes; the exception are TypeLoc nodes, which are passed by-value. We only need to implement the methods for the relevant node types.

4.3 *ASTConsumer:*

ASTConsumer is an interface used to write generic actions on an AST, regardless of how the AST was produced.

4.4 *SourceManager and ASTContext:*

Some of the information about the AST, like source locations and global identifier information, are not stored in the AST nodes themselves, but in the ASTContext and its associated source manager. To retrieve them we need to hand the ASTContext into our RecursiveASTVisitor implementation.

5 DESIGN

The pass is designed in three phases, they are:

5.1 *pass-one:*

In this pass we perform the following functions:

1. Copy the label blocks from inside the function to hoist it outside the immediate function block.
2. Create a list of variables corresponding to each label block that has to be passed as arguments for static scoping $(data\_var.txt)$
3. Copy the body of the structures to hoist them outside the function in which they are declared.

5.2 *pass-two:*

In this pass we perform the following:

1. We remove the body of the label from inside the function
2. Write the arguments that need go be passed to the labels in their definition .The variables have to be passed as a reference to follow static scoping.But in this stage we write the non pointer types ,so as to avoid the error in next stage.
3. We update the function call location with the variables that need to be passed.
4. Remove the structure definition from inside the function body.

5.3 *pass-three:*

In this pass we perform the following functions:

1. We replace the function arguments in definition by the pointer types.
2. In the body of the hoisted functions we replace every use of variables by their pointers.
3. For structure type variables we replace their use in hoisted functions -> as they are pointers now.

A flow sequence of the different phases have been provided in figure 1.

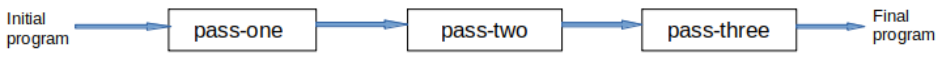

Figure 1. The sequence flow of different phases as mentioned in the makefile

6 A HIGH- LEVEL DESCRIPTION OF THE CHANGES

In this section I will like to give an detailed overview the three passes that I designed.

- **pass-one**
  1. In pass-one I had to copy the label block outside the function body.Here I used the Sourcemanager and Rewriter objects.After getting the start and end of the label block ,I stored it in a string and then appended void ,opening and closing parenthesis.Finally using a Rewriter object wrote the code at the beginning of function.
  2. The second task was to get a list of unique variables for each label block that need to be passed. After observing the ast-dump I figured out that i can get these variables by checking the declaration of the referenced variables in the label blocks and checking whether the declaration lies in the range of the label body.Then wrote the list in a file along with the label name.
  3. A similar procedure was used to hoist the local structures outside then function body.
- **pass-two**
  1. For the task of removing the label from the body ,we just remove the body outermost label.We check this by setting a pointer to the previous label block. If it is null then we delete,else we do not delete.This way we can use the Rewriter object to delete the label inside the function.
  2. Now we read from file the label name along with the list of the variables that need to be passed as arguments and append it to the definition of the function.
  3. At the call location of the label block we pass the address to those variables as that is needed for static scoping. The same Rewriting procedure is used.
- **pass-three**
  1. Now , in this phase we have to replace all the variables used in the label body with their pointers as from the calling location address is being passed .For this every Declrefexpr is checked if their declaration is within the body of label if not then they are replaced by their respective pointers.
  2. The structure type variables have to be handled separately,every use that is undeclared within the block has to be replaced by ->.A similar procedure to above is used to achieve this.
  3. Array type objects have to be handles separately, they do not need a * to access their values.Hence they are not changed.
  4. The function arguments have to changes by their pointers as address is being passed from call location.A simple rewrite will do the job.
- **Data sharing between different phases**
  1. The data between the phases is shared by creating temporary files.That are later removed by the makefile.
  2. Three more files are created for storing the variable lists. These files are read and stored in a vector of a class named varlabel which has two elements , the label name and the variable list.
  3. Some helper functions are used to check whether the function name is present in the vector for various purposes.

7 KNOWN BUGS/ERRORS IN THE IMPLEMENTATION

The following are the some of the known bugs:

1. I have assumed both structs and labels have a globally unique name,though this problem can be easily solved by appending the name of the function along with the struct or label name.
2. Multiple and Multilevel pointers are not supported.

8 ADDITIONAL FEATURES/FUNCTIONALITY

The following are the functionality that works:

- The code works with almost all data types ,I have checked with char,long,double,float,int,string.
- Normal pointer type variable works but advanced use of pointers using address does not work.