Google Summer of Code 2020 Project Proposal

March 2020

Personal Information

Name: Aristotelis Koutsouridis
Address: Anast Katara 1, Acharnes 136 71
City: Athens, Greece (GMT+2)
University: National and Kapodistrian University of Athens
Department: Department of Informatics and Telecommunications
Department Webpage: http://www.di.uoa.gr/eng
Academic email: sdi1600078@di.uoa.gr
Personal email: aristotelis.kouts@gmail.com
Phone Number: +306983479572
Github: https://github.com/arisKoutsou/
LinkedIn: https://www.linkedin.com/in/aristotelis-koutsouridis-0b9789174

Background

I am a 4th year undergraduate student at the Department of Informatics and Telecommunications in Athens. Currently I am working on my bachelor thesis and I will be graduating in September. My thesis focuses on the conversion of .NET stack-based IR to a register-based IR. The goal of this conversion is to perform static analysis on the register-based IR with Doop Framework.
In the past I have been working in a company as a full-stack developer on aquaculture software(June 2018 to November 2019). My most used languages are C/C++ and C#. My C/C++ experience comes from university projects and personal experimentation, whereas my C# experience comes from work.

Last year I took a Compiler course. The course included a project where we had to build a compiler front-end for the mini-java language(a small subset of java, including classes, inheritance, arrays and most of the control flow constructs). The ultimate goal of the project was to generate LLVM IR for mini-java programs. We implemented the semantics checking and code generation by utilizing the visitor pattern, but left out the lexing and parsing for tools like JavaCC and JTB.
Other compiler-related university courses like "Computational Theory" and "Principles of Programming Languages" helped me understand concepts like automata, grammars, Turing machines, language paradigms(logic, functional, lambda calculus).

Project Idea

Extend clang AST to provide information for the type as written in template instantiations.

The goal of this project is to improve compilation diagnostics of the clang compiler frontend. In particular, when instantiating a template, the template argument types may be sugared(e.g. type is a product of 'Typedef', 'Using' statements). In that case we want to have a way of 'remembering' that sugar before substituting the canonicalized template arguments in the template pattern. As a result, any relevant diagnostic exposed to the user will contain the sugared type that was present in the template instantiation(which was written by the user, in contrast to the possible 'typedef' which shall remain abstracted), making it easier for the user to understand diagnostics.

Personal View

I believe that good compiler diagnostics are a major improvement over compiler software, because messages like warnings, errors, notes are the interface of a compiler to the user. They are the most important channel of communication between the compiler software and the average user. Hence, diagnostics should be accurate, intelligent and helpful. In the case of C++ and templates, the mechanism of template argument canonicalization results in inaccurate diagnostics in some cases. For example, when someone declares a variable of type

‘vector<string>’, they expect to see a diagnostic for the type they wrote, not some other alias of the type.

Proposal

Create a new AST Node deriving from ‘Type’ to represent the sugared template type parameter. We may call this node ‘TemplateArgumentSugarType’. An instance of this node will be created when a member access is performed on a template type specialization. When desugaring this node we need to pass the type sugar down the AST hierarchy to the ‘SubstTemplateTypeParmPackType’ and ‘SubstTemplateTypeParmType’ nodes in order to update the ‘SubstTemplateTypeParmPackType.Replaced’ and ‘SubstTemplateTypeParmType.Replaced’ so that they respond to the sugared type. That way we can access the sugared name of the type when printing diagnostics.

Timeline

April

Try to get more familiar with the clang project and understand important concepts for the implementation of the AST Node. Get in contact with mentors and ask questions about the implementation details.

May

Start writing code and testing some ideas. Use the clang API to traverse the AST nodes and get some useful information on the nodes that we are interested in; ‘SubstTemplateTypeParmPackType’, ‘SubstTemplateTypeParmType’, ‘TemplateArgument’. The goal here is to learn how to integrate my solution with the existing code.

June

Implement the new AST Node and link it with existing nodes and try to get diagnostics with sugared type names for simple test cases. For example:

```
template<typename T>
class container {
        T x;
public:
        T get() { return x; }
};

class A { };

typedef A TypedefedA;

int main() {
        container<TypedefedA> v;
        int a = v.get();                    // Expect diagnostic with 'TypedefedA' instead of 'A'.
}
```

July

Find a solution for cases where template argument deduction is used in order to deduce the sugared type of an instance.

August

Start testing the implementation thoroughly and concurrently document the code. Share binaries containing the project solution with friends and colleagues in order to discover more faulty cases, if any. Refactor and polish the final code.

Why LLVM / clang ?

As a programmer I enjoy creating tools that can be easily used by other programmers. A compiler is such a tool, a piece of software that everyone uses daily with ease. I believe GSoC is a great opportunity for me to join the open-source community of llvm and start creating useful tools for people.