Compiler Construction Project - Assignment 3: Types and Variables

October 13, 2023

1 Introduction

In this assignment, you will extend your compiler with a basic type system and simple, register-allocated variable handling. This involves implementing types, variables and scopes. We will elaborate on each of these in the rest of this document.

The assignment is described in Section 2. Section 5 lists what should be handed in to complete this assignment. Also, make sure to update your final report with the new changes. In order to run the framework for assignment 3 you have to run the following command in the build directory: `meson -Dwith-assignment=3 --reconfigure`

2 Assignment

The goal of this assignment is to implement variables, types, scopes, type checking. By the end of this assignment, your compiler should be able to run the snippet of code provided in Listing 1.

```
1         int main() {
2             int8_t a, b;
3             int c, d, e;
4             a = 3;
5             b = 2;
6             c = 12345;
7             d = 54321;
8             if (a > b)
9                 e = c + d;
10            else
11                e = c - d;
12            print e
13        }
14
```

Listing 1: Example of a program to be supported

As you can see your compiler should be able to declare not only one variable but multiple variables. A variable must have a type. And the compiler must check that when a variable is referenced it belongs to the current scope.

2.1 Variables

Many programming languages support variables. In this assignment we consider a variable as a named container for a type of data, in our case integers of different sizes and signedness. In this assignment a variable is always stored in a register however most other programming languages store variables also in memory. The variable name is the usual way to reference the stored value. This separation of name and content allows the name to be used independently of the exact information it represents. Variables are thus bound to a value during run time, and the value of the variable may thus change during the course of program execution. For now we focus on variables stored in registers; this requires an implementation for register allocation. As variables may not be used anymore after a certain line, registers can be cleared to make place for new variables. However, we will restrict ourselves to a naive approach for this assignments, where we assume that the number of variables will never exceed the number of available registers. Furthermore, we will restrict ourselves to solely declaring variables at the start of the scope.

2.2 SymbolTable

An important class that we will use is the symboltable, it allows us to keep track of the variables that we have declared. Bookkeeping which variables are declared is necessary when we start to, for example, generate intermidiate and machine code. In grammarbuilder.h we can see that symboltable is a member variable of class GrammarBuilder. You have to implement some functions of the symboltable class. When you parse your grammar you have to add symbols to the symboltable to retrieve them at a later state of the compilation.

2.3 Types

A variable, besides having a stored value, also has a type. A type tells the compiler how the program represents the value in a variable. Your compiler must be able to directly handle the following types:

C– Type	C++ Type
int	int
unsigned	unsigned int
unsigned int	unsigned int
int8_t	int8_t
uint8_t	uint8_t

In assembly we don't have types, instead, we have registers and memory; we will focus on the former during this assignment. Fundamentally the difference between for example int and int8_t is the size. when we get down to the assembly level an int value would use the entire 32 bits of a register where as int8_t would only need 8 bits. Luckily for us we can use part of a register. 32-bit registers can be used in three ways: As complete 32-bit data registers think of EBX for example. Lower halves of the 32-bit registers can be used as a 16-bit data registers for example BX. Lastly lower and higher halves of the above-mentioned 16-bit register can be used as two 8-bit data registers think of BH & BL for example. Remember right now we are talking about what an int is and what size it has. When we talk about an actual value inside a register we are talking about a variable. Lastly for this assignment, a type tells us how we want to see the data when we call printf.

2.4 Scopes

For this assignment we can think of scopes as a way to encapsulate certain variables. Variables inside a scope cannot be referenced outside of their scope. In other words scopes help us prevent name collision which means we can refer to variables with the same identifier as long as they belong to different scopes.

In grammarbuilder.h we can see that the class GrammarBuilder contains a scope_stack: a stack of scopes where the top element contains the inner most scope that is currently being visited.

2.5 Remarks

The `print` statement should always be used last in a program. This is because `print` uses the C standard library's `printf` under the hood. Which follows the gcc x86 calling convention, which our compiler does not. This leads to our registers being clobbered.

3 Modifying/Extending the framework

If you wish to add **separate '.cpp' files** you will need to add them to the corresponding **meson file**. For example if we look in the folder **src/intermediate-code/src/main/**, we see a **meson.build** file. In this file we have added a path to all the corresponding '.cpp' files we want to compile. Do this if you need to add extra files at any of the compilation steps.

4 Report

You are responsible for proving that a feature works. This means you should hand in (one or more) example program(s), in separate file(s), that use the feature and compile correctly. If the feature concerns optimization, make sure you can toggle the optimization on and off. Then show that with your optimization on, the output code is smaller or more efficient. Describe this and other additional features in your `README`.

We require you to provide a `README` file including any design choices you have made during this assignment. This also includes a summary of the functionalities of each file you have created or modified. Furthermore, the `README` includes a paragraph on what you have learned from this assignment, what the most challenging parts were and how you dealt with these challenges.

5 Submission

This submission is handed in through Brightspace. Go to the course website, and hand in Assignment 3. Beware that the deadline for the Final Report is on the same day.

In Brightspace, hand in

1. a tarball:
   - named `assXgroupY`, with X the assignment number and Y your group number. Name your main folder in the same way (so do not leave it as `assX`).
   - with *all source code* (not only the modified files).
   - **without** the build directory. In general: do not hand in larger submissions than required.
2. the `README` reporting on the assignment.

Failing to adhere to these instructions will result in a penalty to your grade. Please also be aware of the fact that we will not grade work that does not compile. Warnings will result in a penalty in your grade, even if you get warnings when building the framework as is (e.g. unused variable/function warnings).
We use `huisuil` as a reference. So, make sure that your submission compiles on `huisuil`!

You will be graded on the quality of your `README` file, the layout of the code including the modularity and quality of comments, and the functionality of your implementation.

For the date **and time** of the deadline, please check the schedule and submission tab for this assignment on Brightspace.