CS 335 Semester 2023–2024-II: Project Description

28th Feb 2024

Due Your submission is due by Mar 31 2024 11:59 PM IST.

General Policies

- Do not plagiarize or turn in solutions from other sources. You will be PENALIZED if caught.
- Each group will get a total of FOUR FREE DAYS to help with your project submission deadlines. You can use them as and when you want throughout the project. For example, you can submit Milestone 2 one day and ten hours late without penalty, and you will have two free days remaining to potentially utilize for the rest of the semester. The granularity is a day.

Description

The goal of this project is to implement a compilation toolchain, where the input is in Python language and the output is x86_64 code.

Milestone 2

In this milestone, extend your Milestone 1 implementation to support the following features.

- symbol table data structure,
- perform semantic analysis to do limited error checking,
- generate a semantically equivalent 3AC form of the input program that will be used for target code generation during the next milestone,
- include runtime support for function calls.

Maintaining symbol table Extract relevant information from declaration statements and store in the symbol table. Useful information include types of variables and functions along with the source file and line numbers, sizes of variables, and offsets of variables declared within functions.

You can use a hierarchical two-level symbol table structure.

- A global symbol table that maps function names to their local symbol tables,
- A local symbol table for every function that contains the relevant information for the parameters and the local variables of the function.

Feel free to adapt the above structure according to your requirement.

Semantic analysis Your semantic analysis implementation should check for the following error types at a minimum.

- variables are used within the scope of their declarations,
- computations involving variables are type-correct,
- actual and formal arguments of functions match.

Generate 3AC Generate 3-address code (3AC) if the program is syntactically (from Milestone 1) and semantically correct. Remember that you will need to define 3AC instructions corresponding to various Python features allowed in the input program (e.g., `call`, `pushparam`, and `return` where the semantics is obvious from the name).

Maintain the 3AC in an in-memory data structure (e.g., List or Vector). It will be useful for performing optimizations (optional) during code generation.

Runtime support Your implementation of activation records MUST include the following fields.

- space for actual parameters,
- space for return value,
- space for old stack pointers to pop an activation,
- space for saved registers,
- space for locals.

You can optionally add other fields required by your implementation. The exact order of the fields depends on the calling conventions of the target x86_64 architecture [x86 calling conventions, x86 Disassembly/Calling Conventions].

The plan for the next milestone is to generate correct x86_64 assembly from 3AC which can be run (via GAS on Linux). Study the assembly code of a few C programs and refer to x86_64 assembly while designing runtime support for functions.

Output. For correct input programs, your implementation should output the following:

1. A dump of the symbol table of each function as a CSV file. There is no fixed format for the CSV header. The columns should include the syntactic category (token), the lexeme, the type, and the line number in the source at the least.
2. A dump of the 3AC of the functions in text format.

For correct input programs, your implementation should output a text dump of the 3AC of the input Python program with runtime support for activations.

For erroneous input programs, the output should provide meaningful messages that help identify the error that caused the program to be rejected. Do not just print `"Error at line YYY"`. The following are a few examples of acceptable error messages: `"Type mismatch in line 72"`, `"Incompatible operator + with operand of type String"`, and `"Undeclared variable on line 38"`. Here is an *incomplete* list of errors that your implementation should handle:

- All forms of type errors (e.g., array index is not an integer and variables are declared as void type).
- All form of scoping errors.
- Non-context-free restrictions on the language. For example, an array indexed with more indices than its dimension and functions called with incorrect number of parameters.

Submission. Submission will be through BOTH Canvas and CSE Git.

Canvas Create a zip file named "`cs335-project-<groupid>.tar.gz`" and upload it to Canvas. Alternate extensions like `.tar.xz` and `.zip` are allowed. Only one group member should submit the project on behalf of her project mates. The compressed file should contain a folder `milestone2` with the following contents:

- All your source files must be in `milestone2/src` directory.
- Use `Makefile` (or equivalent build tools like `ant`) for your implementation. You are free to use wrapper scripts to automate building *and* executing your compiler.
- Your submission must include a `PDF` file under the `milestone2/doc` directory. The `PDF` file should include compilation and execution instructions. You SHOULD USE LaTeX typesetting system for generating the PDF file.

Git Create a tag called milestone2 BY THE deadline. This is the tag that will be checked out and evaluated by the TAs.

You can create and push tags using the following commands.

```
git tag milestone2 -a
git push origin milestone2
```

Evaluation.

- Your implementation should follow the prescribed steps so that the expected output format is respected.
- We will evaluate your implementations on a Unix-like system, for example, a recent Debian-based distribution.
- We will evaluate the implementations with our inputs and test cases, so remember to test thoroughly.
  - Input test cases will use two spaces for indentation and will have a final newline
  - All input test cases will have an `if __name__ == "__main__":` block
- Our evaluation will mostly focus on correctness, the compilation time is not important.
- The TAs will meet with each group for evaluation. Make sure that your implementation builds and runs correctly.