ASSIGNMENT 4

CS 473: Compiler Design / Fall 2025

Description

In this assignment, your compiler will traverse the AST from previous assignments and translate it to two Intermediate Representation (IR) forms for a stack machine: First a tree, and then a list. Your submission will pass the list IR to the provided interpreter, which can print and execute your programs. You should use the lexical parser that you implemented in Assignment 1, the syntactic parser that you implemented for Assignment 2, and the AST and semantic analysis that you implemented for Assignment 3.

The Cardinal Language

The Cardinal language is defined in a separate document.

Aves Intermediate Representation

The AVES Intermediate Representation is defined in a separate document.

Tree IR

Each IR node has three pointers you can use to translate the AST into a tree form: `tree_ir_1`, `tree_ir_2`, and `tree_ir_3`. For instance, when translating an AST that adds two numbers together, you can use `tree_ir_1` for the left-hand side and `tree_ir_2` for the right-hand side:

AST	`BinOpNode(plus_op, <left>, <right>);`

Tree IR	`ir_node * ir = Ops(ir_add);`
	`ir->tree_ir_1 = /*translate AST <left> to IR */`
	`ir->tree_ir_1 = /*translate AST <right> to IR */`

List IR

Each IR node has a `next` pointer that you can use to turn a tree representation into a list representation. The interpreter provided will expect a list representation. You can turn the example above into a list representation as follows

List IR	

```
ir_node * left  = /*turn ir->tree_ir_1 into a list*/
ir_node * right = /*turn ir->tree_ir_2 into a list*/
// We want to generate the list [left] -> [right] -> Op(ir_add)
// [left] and [right] may be more than one IR instruction
findLast(left)->next = right;  // Generates [left] -> [right]
findLast(right)->next = ir; // Generates [right] -> Op(ir_add)
// We return the list we created above
// starting with the first instruction on the left-hand side
return left;
```

Entry Point

You will reuse files `lexer.lex`, `parser.y` , `semantic_analysis_(symbols|types).[ch]`, and `frame.[ch]` from Assignment 3.

You will modify file `ast_to_ir.c` with a traversal of the AST that transforms it into a Tree IR. You will modify file `ir_tree_to_list.c` with a traversal of the Tree IR that transforms it into a List IR. You can also modify files `transform.c` and `main.c` as you see fit.

You may need to modify your parser and semantic analyses from previous assignments.

Due Date and Resubmission Policy

This assignment has 2 due dates, one for each half of the tests.

First Due Date

The first half of the assignment is due on **November 1st 2025** (Saturday) at **5pm CST**. There is no late policy.

The code and date used for your submission is defined by the last commit to your Git repository.

Second Due Date

The second half of the assignment is due on **November 8th 2025** (Saturday) at **5pm CST**. There is no late policy.

The code and date used for your submission is defined by the last commit to your Git repository.

Resubmissions

Each half of the assignment is resubmitted in separate. ~~To resubmit this assignment, your **original grade** (as defined by the autograder) should be **equal to or higher than 30%** for undergraduate students.~~ You can resubmit the first half of your assignment until **November 8th 2025** at **5pm CST** and the second half until **November 15th 2025** at **5pm CST**. Together with your resubmission, you will have to submit a written description of what you changed from the original submission (on Gradescope).

Note that the first resubmission and the second submission due date are the same!

Bonus Points

This assignment has a total of **30% bonus points**.

You can earn 10% extra points by using Piazza as described in the syllabus. Your posts should be public, tagged with the `assignment4` label, and non-anonymous to the instructors to count towards the bonus. You can claim bonus points through **a Gradescope quiz**.

You can earn 10% extra points by fixing all memory leaks in your program. Each passing test without memory leaks is worth 1% bonus points.

You can earn 10% extra points by passing all the tests by the first due date. Each passing tests 6 through 10 by the first due date is worth a 2% bonus.

Submission and Grading

This assignment is submitted through Github, and has an automatic grade component of 100%. You can check your current grade at any point by submitting your code and checking the autograder. The automatic grade is determined by 10 tests, that will check if your submission outputs the expected result. Each test is worth 10%.

Graduate students

On top of the behavior described above, graduate students have to add support for reading/writing arrays .

Each regular test is only worth 8% for graduate students. The remaining 20% will be graded by providing a 5 minute video showing how you added support for arrays and answering the 4 following questions:

1. How does your compiler reserve room for arrays as global variables?
2. How does your compiler read from arrays? Show the execution of a compiled program that reads from arrays.
3. How does your compiler write to arrays? Show the execution of a compiled program that writes to arrays.
4. Show one limitation of your array implementation.

You can record such a video using Zoom, which you may already have installed to attend lectures remotely. Simply start a meeting (without any other participants), share your screen, and start recording. Note that Zoom requires some time to process your video after you record it, so plan accordingly. Extension requests to upload videos after the due time and date because Zoom is still processing them will be denied.

The maximum length for the video is 5 minutes, instructors will stop watching at the 5 minute mark (nothing past that point in the video will be graded). This video should be a screencast of your IDE open on the code submitted, and you should highlight the code. Note that longer videos are not better videos, and you should record a video as short as needed to show all the expressions and answer the questions above.

The final grade for the assignment will be the grade of the original submission, for assignments without a resubmission; or the average between the original grade and the resubmission grade, for assignments with a resubmission. The grade of the original submission includes any bonus points.

Errors and Omissions

If you find an error or an omission, please post it on Piazza as soon as you find it.

Hardcoding and Academic Integrity

Any hardcoding will result in a 0% grade. Hardcoding is when you submit code that detects which test is being run, and simply outputs the expected result. For instance, detecting that test 22 is running, and replacing the usual execution of your submission with `printf("expected result").`

The academic integrity policy described in the syllabus applies to this assignment. You are responsible for writing all the code that you submit. We will use an automatic tool that detects plagiarism on all submitted code, and we will investigate all instances where plagiarism is more than likely.

Please refer to the syllabus for the full academic integrity policy.