Compiler Project
PA5 – Final Submission
Due: 2024-04-30 11:59pm

The final submission of the project adds no new functionality, but provides an opportunity to correct PA1-PA4 errors in your compiler and make improvements as you wish. It also provides an opportunity to add extensions to your compiler for extra credit. The extra credit is not needed – all grades can be obtained with just the basic project unless you are on a team.

You need to submit a guide for your compiler, preferably as a PDF. Turn in all project files on the "PA5—Final Compiler Submission" assignment on Gradescope.

-=-=-=-=-=-=-=-=-=-=-

REQUIRED ITEMS: GUIDE, ASTChanges.txt, SOURCE CODE

Recall how x64 documentation was difficult to read. Try not to repeat those mistakes in your guide. You do not have to restate the obvious. Assume your reader is generally familiar with compilers, doesn't need to know what is an AST, and doesn't need to know the specifics of bytecode generation. Don't restate what you already did in ASTChanges.txt.

YOUR GUIDE IS NOT A JUSTIFICATION FOR DESIGN DECISIONS

Instead, it is a guide. In as **few sentences** as possible, concisely describe how you handled the parts of the compiler.

1) Syntactic Analysis- Did you use recursive descent or a PDA? Did you make everything a token, or minimize the number of tokens? Anything else?
2) AST Generation- "We assume general familiarity with ASTs. See ASTChanges.txt for data beyond syntax that is stored in ASTs."
3) Contextual Analysis- Did you do one traversal or two? Where is SI done?
4) Code Generation- Did you do optimization? How is your memory laid out?
5) Finally: Any greedy decisions to watch out for? For example, if you made everything 64-bit even though JLS requires int to be 32-bit, then how did you make sure add/multiply/subtract did not cause issues? If there are indeed parts of your compiler where you simply did not handle errors, that is actually fine, but mention those errors so that someone else can take over your Compiler.

There is no minimum page count, paragraph count, word count, etc. Your guide could even entirely be drawn in MSPaint. The most important part is that it conveys how your compiler is organized. Please do not submit more than a page.

If you have a GitHub, a good idea is to do documentation in the README.md file.

OPTIONAL ITEMS: EXTRA CREDIT

Assume PA1 – PA5 grades are normalized to 100 points. With the Compiler worth 60% of your grade, each extra credit point is worth 0.6% of your course grade.

Each extra credit item has requirements. Some require you to supply tests that can check the proper functionality of any additional implementation. These tests do not have to be exhaustive, but please provide at least two input files (one fail, one pass).

Some extra credit items do not have to be implemented in all parts of the compiler. For example, the push-down automata is only a PA1 constraint and does not require redoing the rest of the compiler (PA2-PA4). If an extra credit item requires multiple PA checkpoints to be reached, you are allowed to partially complete it for partial credit, but only where it makes sense to award partial credit (no trivial additions).

There are many extra credit items that are not listed here. You can come up with your own extra credit options, but please provide how many points you think that item should be worth. (It may be worthwhile to ask the course staff for such).

Pts	Tests	PAs	Description
1	Y	All	Allow initialization expressions for static fields.
1	Y	All	Parameterized class constructions. Only one constructor per class unless method overloading.
2	Y	All	For loops. Specify the new Grammar in your PA5.
2	Y	PA4	Do both: Short circuit `&&` and `||` expressions, where "`FALSE && ...`" will not evaluate subsequent expressions and "`TRUE || ...`" will not evaluate subsequent expressions.
4	Y	All	Implement String. Note, "String.`length`" must resolve to a proper variable, and "double quoted string data" needs to be parsed. You do not need to implement BinExpr on Strings.
1	Y	PA3-4	Fix `System.out.println` to allow for a String parameter.
1	Y	PA3-4	Implement ".`length`" for arrays in PA3 and PA4.
2-4	N	PA1	Implement PA1 with a PDA (and optionally PA2)
1-5	N	PA4	Try to minimize the register usage of PA4.
2	Y	PA2-3	Implement method overloading (signature is by parameter list, not return type).
1-2	Y	PA4	Implement method overloading in PA4 as well, and an additional point for overloading constructors.
3-15	Y	All	Enable `instanceof` and `super`, and allow classes to `extend` other classes. Make sure type-checking is extended appropriately and ensure all methods are virtual methods.
1	N	PA3	Do a single traversal for Identification and Type-Checking
*	N	PA4	Apply some optimization algorithms. Ensure you specify how many points you think it should be worth (amount awarded is not guaranteed).

More Extra Credit Ideas

Pts	Tests	PAs	Description
1-3	Y	PA4	Come up with a secret handshake (a specific consecutive set of Statements) that does no meaningful computation, but if detected by your compiler, your compiler will then do something special. For example, swap all multiply and addition operations. Or, when storing data in an (Ix)AssignStmt, encode data with an XOR, and decode when reading data (that way original values are only seen in registers, but the memory always looks corrupted). Or simply output something on the screen that isn't a part of the normal code.
1	N	PA3	In contextual analysis, you required a return statement at the end of every non-void method. Extend this functionality to ensure all code PATHS end with a return statement instead for non-void methods.
1	N	PA4	Extend ModRMSIB (if you haven't already) to ensure proper use of the values for mod=00 and 01. This means only writing zero/one byte when you otherwise pick mod=10 and greedily always output 4 bytes.
5-20	N	All	Enable the use of shared libraries. This will require you to either redo the ELFMaker entirely or cleverly add on imports via PLT. Additional points come from properly implementing bss. For that, you will need to read into the GOT/PLT sections to find where is "bss" during runtime.
*	Y	PA1-3 PA4	Try implementing features in other programming languages. How about a foreach loop using the .length parameter? Something more difficult could be adding operator overloading from C++. Specify how many points you think such an extension is worth, and whether you implemented it in PA4 as well.
1	Y	PA1-3	Add support for float.
1-3	Y	PA4	Add support for float in PA4.
1	Y	All	Add support for char. Make sure you can parse single quotes.
1	N	PA4	Every executable that is generated by your compiler has a special signature hidden somewhere. Just a blob of binary data that is never accessed or written to by the input source code. This will act as a fingerprint where you can trace who used your compiler for binaries.
1-3	N	PA4	Be evil and use CPUID to detect the manufacturer string of whatever is running your executable. If it isn't a wanted CPU, do evil things like unoptimized code or random pointless loops injected into the code.
4-5	Y	PA4	Try targeting MIPS (If you are taking COMP-541, target only the subset to see if you can get it to run on the emulator).
3-8	Y	PA4	Change your ELFMaker to something else, see if you can get your program to run Windows, an M1+ processor, etc.
2-4	N	PA4	Add the ability to automatically garbage collect objects that are no longer referenced (sys_munmap).
1-3	Y	All	Create some intricate code (e.g., Sieve of Eratosthenes, Shortest-path, etc.) that runs in miniJava.

Make sure to submit this on GradeScope.