CS 415: Compilers
Spring 2022
Project 2: A Simple Compiler Front-End
Due date: Friday, April 15

Sample Language

Using flex and bison, you are to write a parser and code generator for our sample language. There are no procedures in this language. Base types are limited to integer and boolean. Composit types are limited to single dimensional arrays of base types, indexed by integers. Only the following statements are included: for iterations, while loops, if-then-else, assignment, print, and compound. Operators are restricted to arithmetic, logical, and relational. The context-free grammar of our sample language can be found here .

Front-End: Project Description

As part of this project, you will given the full scanner (scan.l) and a skeleton parser (parse.y), together with a nearly complete implementation of the symbol table (symtab.h and symtab.c). Funtions needed to generate ILOC code are also provided (instrutils.h and instrutils.c). As part of this project, you have to write a syntax-directed translation scheme. You will have to

- Augment your parser to perform semantic analysis, in particular type checking. Your type checker has to detect and report at least the following semantic errors (error productions for syntactic errors are already included in the provided code):
  - Declarations
    "\n*Error: duplicate declaration of %s\n"**
  - If stmt
    "\n*Error: exp in if stmt must be boolean\n"**
  - While stmt:
    "\n*Error: exp in while stmt must be boolean\n"**
  - Assignment stmt
    "\n*Error: assignment types do not match\n"**
    "\n*Error: assignment to whole array\n"**
  - Array subscript operation id[exp] in exp
    "\n*Error: subscript exp not type integer\n"**
    "\n*Error: id %s is not an array\n"**
  - for statement
    "\n*Error: lower bound exceeds upper bound\n"**
    "\n*Error: induction variable not scalar integer variable\n"**
  - Identifier
    "\n*Error: undeclared identifier %s\n"**
  - Arithmetic, logical, and relational operators in exp
    "\n*Error: types of operands for operation %s do not match\n";**
- Your compiler will generate ILOC code that conforms to the register-register model. Our language does not allow for aliasing. You should test the correctness of your generated ILOC code by running it on our ILOC simulator on the ilab cluster.
  Our example language does not have procedure calls. Therefore, all variables are statically allocated, i.e., there is no need for activation records. The static area starts at memory location 1024. Addresses lower than 1024 are reserved for register spilling. This convention will allow you to run your project 1 instruction scheduler on each basic block of your generated code. The register **r0** must always contain the starting address (namely 1024) of the static area.

Specific Details

Names are **case sensitive** .

Logical operators (and, or) require boolean arguments. The arithmetic and relational operators (e.g., +, <=) require integer arguments. Integer and boolean arguments cannot be mixed. The equality/inequality operators (==, !=) apply to both types, as long as the two argument types match.

- You may only use the following ILOC instructions. All these instructions are implemented in our ILOC simulator.
  - no operation: ***nop*** .
  - arithemtic: ***add, sub, mult*** .
  - logical: ***and, or*** .
  - memory: ***load, loadI, loadAO, loadAI, store, storeAO, storeAI*** .
  - control flow: ***br, cbr, cmp_LT, cmp_LE, cmp_EQ, cmp_NE*** .
  - I/O: ***outputAI*** .

  Please see files *instrutil.h* and *instrutil.c* for details about the required formats. Procedures **emit, NextRegister, and NextLabel** are also defined within these files.
- You may want to generate ***nop*** instructions as targets of branches and conditional branches, e.g., ***L1: nop*** .
- Boolean values are represented as 4 bytes entities with 0 == FALSE and 1 == TRUE. Therefore, regular load and store instruction can be used.
- The evaluation of an *exp* will always result in an integer or boolean value, while the evaluation of an *condexp* will always result in a boolean (0 or 1) value. An ILOC ***cmp_*** instruction writes a boolean value into its target register.
- The function *NextLabel* will generate a new (fresh) label each time it is called.

Getting Started

The following code is provided as a **starting point** for your project. The files listed below live on the ilab cluster in subdirectory ***~uli/cs415/projects/proj2/students*** . Please copy the files into your subdirectory of choice using the command
cp -r ~uli/cs415/projects/proj2/students myProjectSubdirectory .
You also may download the proj2.tar file. You should modify files for your project according to the followin guideline.

1. Declaration of types and attribute operations: attr.h and attr.c
2. Additional information needed in symbol table and symbol table operations: symtab.h and symtab.c,
3. Scanner: scan.l (flex); no modification
4. Parser: parse.y (bison); here is most of your work. The current version contains some valuable hints of how your code should look like.
5. Makefile; no modification
6. There are several test programs in subdirectory "testcases"

You can generate an executable called **codegen** by typing *make* . Your compiler expects the input on stdin, i.e., you can call the compiler on the provided demo1 input program as follows: *./codegen < testcases/demo1* .

A sample solution compiler (executable) is also available, called **sol-codegen**. I suggest to use a soft link to this sample solution in your project directory: *ln -s ~uli/cs415/projects/proj2/sol-codegen .* .

Due Date

The project is due at **11:59 pm on Friday April 15, 2022**--that is, one minute before midnight.

If you have specific, overriding, personal reasons why these dates are unreasonable, you should discuss them directly with the instructor **before** the deadline.

A late penalty of 20% will be assessed for every starting 24 hour period the submission is late.

Questions and Submission Procedure

All questions related to the project should be posted on piazza. The submission will be through canvas. Please see submission instructions as posted on canvas. DO NOT SUBMIT EXECUTABLES!

Grading Criteria

The project will be mainly graded on functionality. You will receive no credit for the entire project if your code does not compile and run on the **ilab cluster** . We will use automatic testing tools, so make sure that your error messages are exactly as described above.