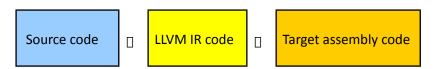
CMPE 230 Computer Systems Spring 2021

Project (due April 26th)

(This project can be implemented in groups of at most two students. You can use C/C++ or Java)

LLVM (low-level virtual machine - http://llvm.org/) is a compiler infrastructure that provides IR (intermediate representation) that can be used to generate code for various target architectures. We will use LLVM version 3.3 in this project. You can download the pre-built clang and llvm binaries for Ubuntu from this link: https://releases.llvm.org/download.html#3.3



In this project, you will develop a translator called mylang2IR for a language called Mylang. Mylang language statements will be as follows:

- 1. Variables will be integer variables. Their default value will be 0 (i.e. if they are not initialized).
- 2. Executable statements will consist of one-line statements, while-loop, and if compound statements. Note that no nested while-loops or if statements are allowed.
- 3. One-line statements are either assignment statements or print statements that print the value of an expression.
- 4. There is a function choose (expr1, expr2, expr3, expr4) which returns expr2 if expr1 is equal to 0, returns expr3 if expr1 is positive and returns expr4 if expr1 is negative.
- 5. As operations in expressions, you are required to implement only multiplication, division addition, and subtraction: *, /, +, . These are binary operand operations. Unary minus operation is *not* supported but parentheses are allowed. Operator precedence needs to be implemented as in other programming languages, such as C or Java.
- 6. On a line, everything after the # sign is considered as comments.
- 7. If statement will have the following format:

```
if (expr) {
    .....
}
```

If expr has a nonzero value, it means true. If expr has zero value, it means false. Note that there is no else part. There are no nested if statements.

8. While loop will have the following format:

```
while (expr) {
    .....
}
```

If expr has a nonzero value, it means true. If expr has zero value, it means false. There are no nested while statements.

- 9. print(id) statement, prints the value of variable id.
- 10. In case of a syntax error in line x, print "Line x: syntax error"

mylang2IR will generate low-level LLVM IR code that will compute and output these statements.

For example, given the following code in file.my:

```
n = 10
f0 = 0
f1 = 1
i = 2
while(n) {
t = f1
```

```
f1 = f0 +f1
print(f1)
f0 = t
n = n - 1
}
```

mylang2IR will output the following IR code:

```
LLVM IR (file.II)
; ModuleID = 'mylang2ir'
declare i32 @printf(i8*, ...)
@print.str = constant [4 x i8] c"%d\0A\00"
define i32 @main() {
  %n = alloca i32
  %f0 = alloca i32
  %f1 = alloca i32
  %i = alloca i32
  %t = alloca i32
  store i32 0, i32* %n
  store i32 0, i32* %f0
  store i32 0, i32* %f1
  store i32 0, i32* %i
  store i32 0, i32* %t
  store i32 10, i32* %n
  store i32 0, i32* %f0
 store i32 1, i32* %f1
store i32 2, i32* %i
 br label %whcond
whoond:
  %t1 = load i32* %n
%t2 = icmp ne i32 %t1, 0
 br i1 %t2, label %whbody, label %whend
  %t3 = load i32* %f1
  store i32 %t3, i32* %t
  %t4 = load i32* %f0
  %t5 = load i32* %f1
  %t6 = add i32 %t4, %t5
  store i32 %t6, i32* %f1
  %t7 = load i32* %f1
                         @printf(i8* getelementptr ([4 x i8]* @print.str, i32 0, i32 0), i32 \%t7 )
  call i32 (i8*, ...)
%t8 = load i32* %t
  store i32 %t8, i32* %f0
      = load i32* %n
  %t10 = sub i32 %t9, 1
  store i32 %t10, i32* %n
 br label %whcond
  ret i32 0
```

Please note the following about the IR code:

- LLVM IR uses static single assignment (SSA) based representation. In assignment statements, variables are assigned a value once.
- alloca is used to allocate space for variables and return the address of the allocation.
- In IR, variables start with the % sign.
- The keywords i8, i16, and i32 mean 8-bit, 16-bit, and 32-bit type respectively.
- The keyword i32* means 32-bit pointer.
- Variables %ti (where i is an integer) are temporary variables.
- The yellow-colored code defines the module name and the prototype for the printf output statement. Generate this part as it is shown in the above example.
- The green-colored code is for printing the value of a variable using the printf function.
- You can assume only binary operations (+, -, *, /) will be used in the expressions. All variables and operations are integer operations. Division operation gives the quotient.

Commands	Explanation
mylang2ir file.my	Runs mylang2ir on file.my and
	produces IR code in file.ll
lli file.ll	Runs the Ilvm interpreter & dynamic
	compiler. For the above example, this
	command produces the output:
	1
	2
	3
	5
	8
	13
	21
	34
	55
	89
llc file.ll -o file.s	Invokes IIc compiler to produce
	assembler code
clang file.s -o file.exe	Compiles assembler code to produce
	the executable
./file.exe	Runs the executable. For the above
	example, this command produces the
	output:
	1
	2
	3
	5
	8
	13
	21
	34
	55
	89

Note that you are implementing only the mylang2ir program. The others are LLVM commands. You should prepare a makefile that compiles your source code and generates the executable mylang2ir.

Grading

Your project will be graded according to the following criteria:

Documentation (a written document describing	g 12%
how you implemented your project)	
Comments in your code	8%
Implementation and tests	80%

Late Submission

If the project is submitted late, the following penalties will be applied:

0 < hours late <= 24 : 25%24 < hours late <= 48 : 50%

• hours late > 48: 100%

Timestamping

The project file should include your names in it. Please timestamp your project file using https://opentimestamps.org/ before you submit it. Keep the project file and its corresponding timestamp .ots file.