Learning LLVM

work in progress

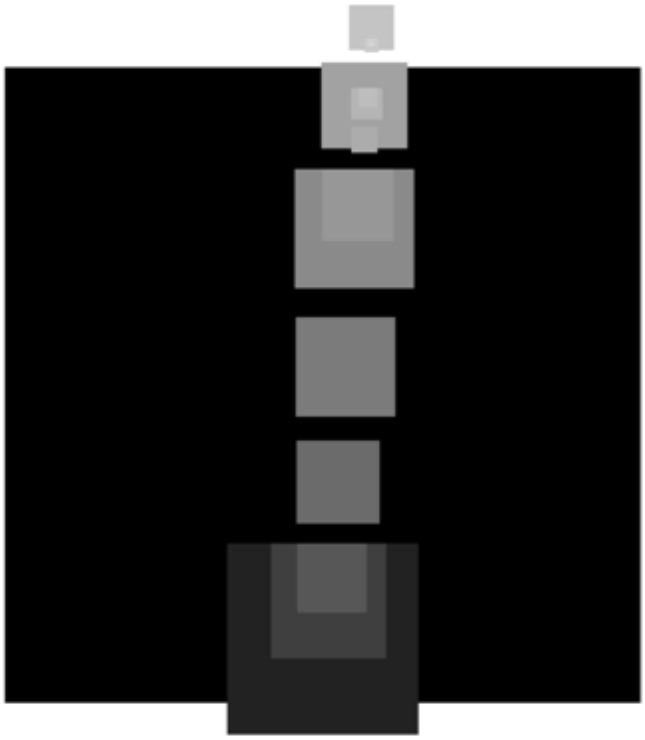

Table of Contents

Installing the Toolchain

OS X

- XCode
- Homebrew
- LLVM

From C

After googling “Hello World in LLVM” I was presented with several results, but they had one thing in common - They all proposed that the best way to learn how hello-world works in LLVM is by compiling a C version of hello-world with llvm-gcc and then looking at the output:

hello_llvm.c

```
#include <stdio.h>
int main( )
{
  printf("Hello World!\n");
}
```

```
> llvm-gcc % -S -emit-llvm
```

hello_llvm.s

```
; ModuleID = 'hello_llvm.c'
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-i64:64:64-f32:32:32-f64:64:64
target triple = "x86_64-apple-darwin11.4"

@.str = private constant [13 x i8] c"Hello World!\00", align 1 ; <[13 x i8]*> [#uses=1]

define i32 @main() nounwind ssp {
entry:
  %retval = alloca i32                              ; <i32*> [#uses=1]
  %"alloca point" = bitcast i32 0 to i32            ; <i32> [#uses=0]
  %0 = call i32 @puts(i8* getelementptr inbounds ([13 x i8]* @.str, i64 0, i64 0)) nounwind ; <i3
  br label %return

return:                                             ; preds = %entry
  %retval1 = load i32* %retval                      ; <i32> [#uses=1]
  ret i32 %retval1
}

declare i32 @puts(i8*)
```

```
> lli hello_llvm.s
```

```
Hello World
```

It works!!! But... The code is pretty cluttered.

Exploring the Intermediate Representation

When presented with a mess like this, the first thing I always try to do is see what I can remove...

I figure the first three lines can get the hell outta here!!

```
; ModuleID = 'hello_llvm.c'
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-i64:64:64-f32:32:32-f64:64:64
target triple = "x86_64-apple-darwin11.4"
```

Everything compiles and runs, but just to be sure I look up what those lines were doing anyway:

- `;` is a comment - everything following on the same line is ignored
- `target datalayout` sets the format of various bits of data
- `target triple` specifies the operationg environment that the program is targeting

The targets seem to have reasonable defaults and aren't required so away they go. With them gone let's have a look at what remains...

The constant declaration of "Hello World" looks like a good place to start:

```
@.str = private constant [13 x i8] c"Hello World!\00", align 1 ; <[13 x i8]*> [#uses=1]
```

Let's modify it to say goodbye and see what happenes...

```
@.str = private constant [13 x i8] c"Goodbye World!\00", align 1 ; <[13 x i8]*> [#uses=1]
```

Error!

```
lli: hello_llvm.s:2:36: error: constant expression type mismatch
@.str = private constant [13 x i8] c"Goodbye World!\00", align 1 ; <[13 x i8]*> [#uses=1]
                                   ^
```

It looks like the length declared has to match the string length otherwise boom. So we modify the declaration

```
@.str = private constant [15 x i8] c"Goodbye World!\00", align 1 ; <[13 x i8]*> [#uses=1]
```

Error!

```
lli: hello_llvm.s:8:62: error: '@.str' defined with type '[15 x i8]*'
  %0 = call i32 @puts(i8* getelementptr inbounds ([13 x i8]* @.str, i64 0, i64 0)) nounwind ; <i3
                                                                 ^
```

So... references are typed too?

Let's update that.

```
  %0 = call i32 @puts(i8* getelementptr inbounds ([15 x i8]* @.str, i64 0, i64 0)) nounwind ; <i3
```

And...

```
Goodbye World!
```

Woot!

Let's have another look at our program

```
@.str = private constant [15 x i8] c"Goodbye World!\00", align 1 ; <[13 x i8]*> [#uses=1]

define i32 @main() nounwind ssp {
entry:
  %retval = alloca i32                            ; <i32*> [#uses=1]
  %"alloca point" = bitcast i32 0 to i32          ; <i32> [#uses=0]
  %0 = call i32 @puts(i8* getelementptr inbounds ([15 x i8]* @.str, i64 0, i64 0)) nounwind ; <i3
  br label %return

return:                                           ; preds = %entry
  %retval1 = load i32* %retval                    ; <i32> [#uses=1]
  ret i32 %retval1
}

declare i32 @puts(i8*)
```

According to IBM the essential components of an LLVM IR program are...

- Comments
- Global identifiers
- Local identifiers
- A strong type system
- Vectors or Arrays
- Global string constants
- Functions

- Return statements
- Function calls

Let's see if we can change the name of our @.str constant to @greeting:

```
@greeting = private constant [15 x i8] c"Goodbye World!\00", align 1 ; <[13 x i8]*> [#uses=1]

define i32 @main() nounwind ssp {
entry:
  %retval = alloca i32                               ; <i32*> [#uses=1]
  %"alloca point" = bitcast i32 0 to i32             ; <i32> [#uses=0]
  %0 = call i32 @puts(i8* getelementptr inbounds ([15 x i8]* @greeting, i64 0, i64 0)) nounwind
  br label %return

return:                                              ; preds = %entry
  %retval1 = load i32* %retval                       ; <i32> [#uses=1]
  ret i32 %retval1
}

declare i32 @puts(i8*)
```

And running...

```
Goodbye World!
```

Yay!

Super Small:

```
@greeting = constant [15 x i8] c"Goodbye World!\00"

define i32 @main() nounwind ssp {
entry:
  %0 = call i32 @puts(i8* getelementptr inbounds ([15 x i8]* @greeting, i64 0, i64 0)) nounwind
  ret i32 0
}

declare i32 @puts(i8*)
```

Appendix

- http://www.ibm.com/developerworks/library/os-createcompilerllvm1/
- http://www.yellosoft.us/hello-llvm
- http://llvm.org/docs/WritingAnLLVMPass.html