Use Mac OS X for example

Before installation: Make sure enough free space like 16 GB or 30 GB. Command line works well. Make sure the minimum required Cmake version 3.4.3 is installed. Check your current Cmake version using command "cmake --version". If not, download and install it.

1. Installation:

Create your own work directory:

```
$ mkdir mywork
$ cd mywork
```

Get LLVM:

```
$ svn co http://llvm.org/svn/llvm-project/llvm/trunk llvm
```

Get Clang:

```
$ cd llvm/tools
$ svn co http://llvm.org/svn/llvm-project/cfe/trunk clang
$ cd ../..
```

Build LLVM and Clang:

```
$ mkdir build
$ cd build
$ cmake -G "Unix Makefiles" ../llvm
$ make
```

Add "llvm/build/bin/" to your PATH. Now "$ clang --version" shows your Clang and LLVM version indicating LLVM should work.

The installation is a simplified version of Clang installation documentation. See the original documentation and install extra files according to your project.

Notes: To get LLVM and Clang, we can also download the .tar file and extract them in proper directory tree which will be much faster than svn. Svn took a couple of minutes and the building process took me 1 hour and 40 minutes.

2. Basic Usage of LLVM and Clang:

Compiling (similar to GCC):

Copy the text below to create Fibonacci.c but it is free to create your own little program to test basic usage.

```
#include <stdio.h>
```

```
int main()
{
   int n, first = 0, second = 1, next, c;

   n = 10;
   printf("First %d terms of Fibonacci series are :-\n",n);

   for ( c = 0 ; c < n ; c++ )
   {
      if ( c <= 1 )
         next = c;
      else
      {
         next = first + second;
         first = second;
         second = next;
      }
      printf("%d\n",next);
   }

   return 0;
}
```

```
$ clang Fibonacci.c -o Fibo
(If header file not found, find your std lib directory and make it C_INCLUDE_PATH)
$ ./Fibo
```

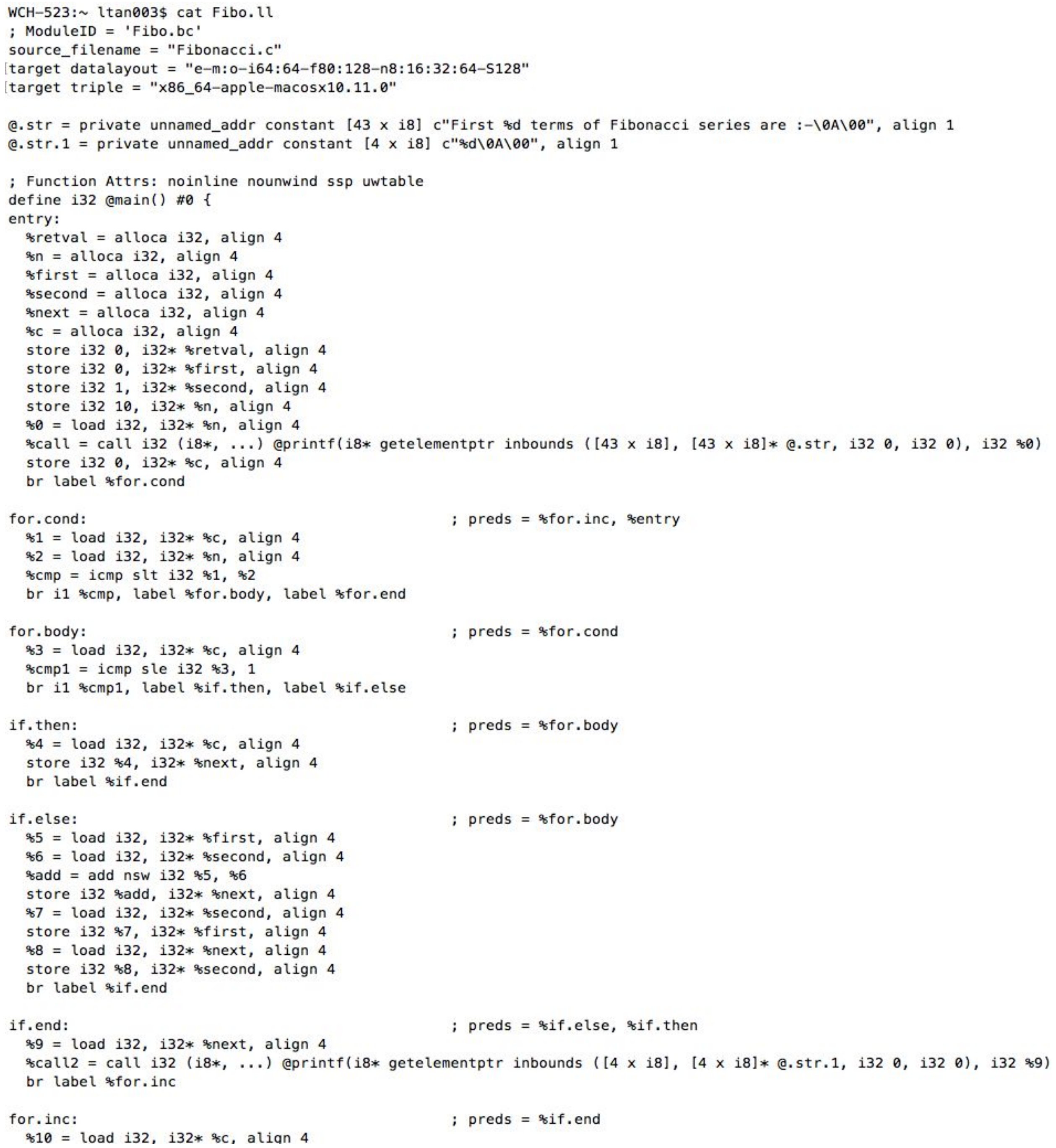
Generating bitcode:

```
$ clang -emit-llvm -O0 -c Fibonacci.c -o Fibo.bc        %generate bitcode Fibo.bc
$ llc Fibo.bc                                           %generate assembly Fibo.s
$ gcc Fibo.s -o Fibo                                    %generate executable Fibo
$ ./Fibo                                                %run Fibo
```

Generating human readable LLVM assembly

```
$ llvm-dis -f Fibo.bc                                   %generate readable assembly Fibo.ll
$ cat Fibo.ll
```

Example result: (partial)

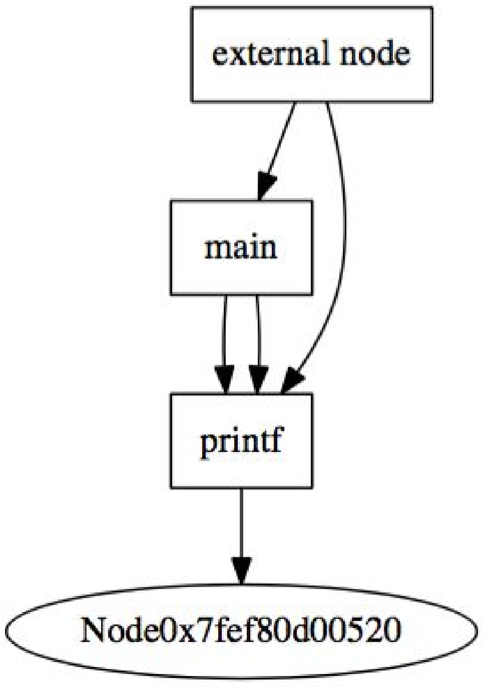


```
WCH-523:~ ltan003$ cat Fibo.ll
; ModuleID = 'Fibo.bc'
source_filename = "Fibonacci.c"
target datalayout = "e-m:o-i64:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx10.11.0"

@.str = private unnamed_addr constant [43 x i8] c"First %d terms of Fibonacci series are :-\0A\00", align 1
@.str.1 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1

; Function Attrs: noinline nounwind ssp uwtable
define i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %n = alloca i32, align 4
  %first = alloca i32, align 4
  %second = alloca i32, align 4
  %next = alloca i32, align 4
  %c = alloca i32, align 4
  store i32 0, i32* %retval, align 4
  store i32 0, i32* %first, align 4
  store i32 1, i32* %second, align 4
  store i32 10, i32* %n, align 4
  %0 = load i32, i32* %n, align 4
  %call = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([43 x i8], [43 x i8]* @.str, i32 0, i32 0), i32 %0)
  store i32 0, i32* %c, align 4
  br label %for.cond

for.cond:                                         ; preds = %for.inc, %entry
  %1 = load i32, i32* %c, align 4
  %2 = load i32, i32* %n, align 4
  %cmp = icmp slt i32 %1, %2
  br i1 %cmp, label %for.body, label %for.end

for.body:                                         ; preds = %for.cond
  %3 = load i32, i32* %c, align 4
  %cmp1 = icmp sle i32 %3, 1
  br i1 %cmp1, label %if.then, label %if.else

if.then:                                          ; preds = %for.body
  %4 = load i32, i32* %c, align 4
  store i32 %4, i32* %next, align 4
  br label %if.end

if.else:                                          ; preds = %for.body
  %5 = load i32, i32* %first, align 4
  %6 = load i32, i32* %second, align 4
  %add = add nsw i32 %5, %6
  store i32 %add, i32* %next, align 4
  %7 = load i32, i32* %second, align 4
  store i32 %7, i32* %first, align 4
  %8 = load i32, i32* %next, align 4
  store i32 %8, i32* %second, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %9 = load i32, i32* %next, align 4
  %call2 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @.str.1, i32 0, i32 0), i32 %9)
  br label %for.inc

for.inc:                                          ; preds = %if.end
  %10 = load i32, i32* %c, align 4
```

Analyze using graphs:
Example here is the call graph:

```
$ opt -dot-callgraph Fibo.bc                    %generate callgraph.dot for Fibo.bc
$ dot -Tps callgraph.dot -o outfile.ps          %convert dot to image
```

Open the outfile.ps to see the call graph of the program Fibonacci.c.



Call graph

3. Passes:

Analysis passes:
-basiccg: Basic CallGraph Construction; Source code

-da: Dependence Analysis; Source code

-domtree: Dominator Tree Construction; Source code

-loops: Natural Loop Information; Source code

Transform passes:
-bb-vectorize: Basic-Block Vectorization; Source code

-constmerge: Merge Duplicate Global Constants; Source code

-gvn: Global Value Numbering; Source code

-loop-extract: Extract loops into new functions; Source code

-loop-unroll: Unroll loops; Source code

-memcpyopt: MemCpy Optimization; Source code

Or other interesting passes listed in the documentation which also shows description of the above passes.

4. Writing a pass

Please refer to my lecture slides
https://docs.google.com/presentation/d/1RwSz9XT30UfE0AI9udfyY5uEHFH7qMRQnugRPISVUj4/edit?usp=sharing

You may find my reference material on the first page of my slides helpful.

Below are materials I referred when I write the documentation. Hope they will be helpful.
Installation of LLVM and Clang on Windows machine
Installation on Mac OS X
LLVM for Grad Students
Using LLVM for Program Transformation