# Assignment

First Parabix example, assignment for this week.

As a first example of Parabix programming, we will look at the hexlines utility.

The utility is found in parabix-devel/tools/util/hexlines.cpp

The purpose of hexlines is to print out the contents of each line of a file

in hexadecimal notation.

Method

1. The file is read as a ByteStream and transposed using the S2P\_Kernel

to the parallel bit stream representation BasisBits.

For i in 0..7:

BasisBits[i] is the bit stream consisting of bit i of each byte of ByteStream.

See line 96 of hexlines.cpp

2. A stream of 1 bits is created to mark every position that is not a LineFeed.

These are the positions that need to be converted to hexadecimal.

This is carried out by applying the CharacterClassKernelBuilder to the

character class consisting of all bytes except 0x0A. The result is the

nonLF stream (line 103).

3. For every position to be converted to hex, we will generate two bytes

instead of one, being the ASCII characters for the high and the low 4 bits

of the byte. This means that we need to create longer streams, with

one extra position inserted for every 1 bit in the nonLF stream.

We use a spread mask for this purpose. The SpreadByMask kernel will

spread out the bits of source streams to the positions marked by one

bits in the mask.

Suppose we have a file of two lines, one of length 4 and one of length 3

(excluding the final LF in each case). In this case, the nonLF stream is

nonLF = 1111.111.

Here we use "." to denote 0 bits, so that the 1 bits stand out.

The spread mask that we want has nine 1 bits, one for each source position

(including the LFs). The mask we want is thus:

hexInsertMask = 1.1.1.1.11.1.1.1

This mask is determined by the UnitInsertionSpreadMask kernel.

4. The SpreadByMask kernel is then applied to spread out the bits of each

of the 8 basis streams to give a new (and longer) basis set: spreadBasis

5. The next step is to replace the two positions of each original data byte

by the hexadecimal codes. This is carried out by the Hexify kernel

written in Pablo.

6. Finally the output of Pablo is transposed back to a byte stream form and

written to stdout.

Exercise:

(1) Explain the output that you see with the following commands:

1. bin/hexlines ../QA/testfiles/simple1

(b) bin/hexlines ../QA/testfiles/simple1 -ShowPablo

(c) bin/hexlines ../QA/testfiles/simple1 -trace=Byte

(2) How would you modify the program to generate lower case hex letters rather than

upper case letters?

# Compile your first parabix program

* Step1 Install cmake, llvm, boost

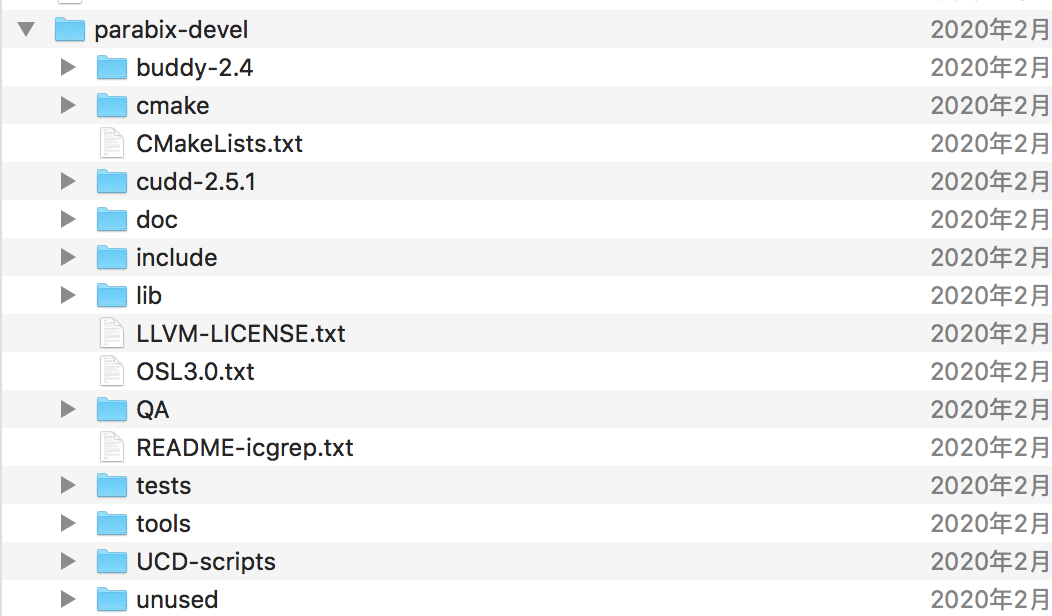
brew install cmake

brew install llvm

brew install boost

* Step2 Compile the source code

Go to the parabix-dev directory



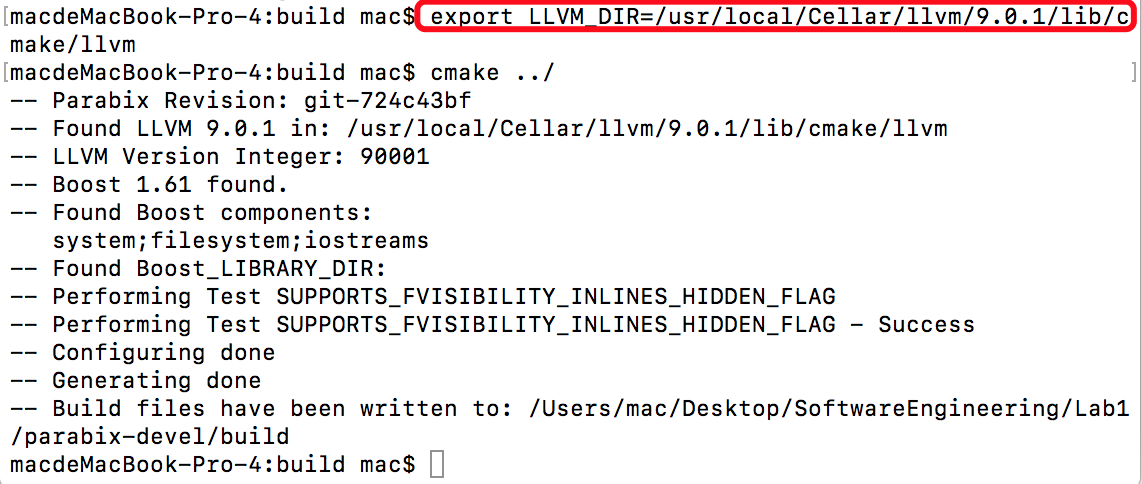
Type in the following command:

mkdir build

cd build

export LLVM\_DIR= Path where llvm is installed (in my mac it is /usr/local/Cellar/llvm/9.0.1/lib/cmake/llvm)

cmake ../



make

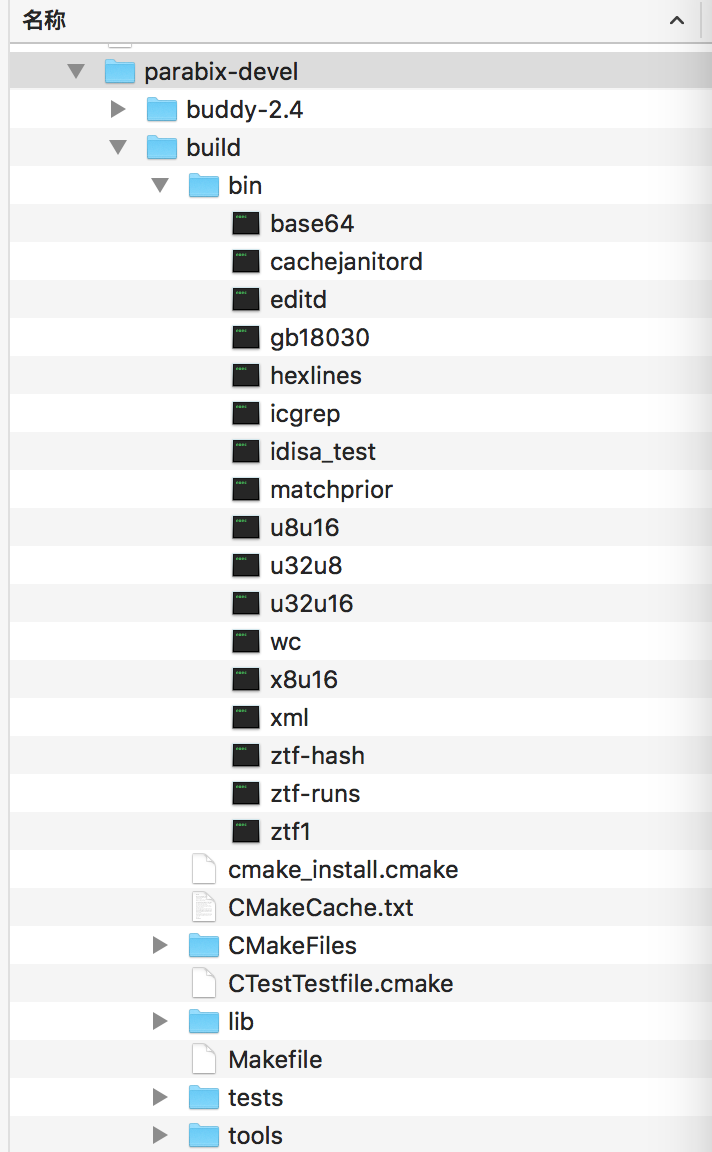


Figure 1 Your directory should look like this after “make”

* Run the program by typing in the following command and explain the output

make greptest

bin/hexlines ../QA/testfiles/simple1

bin/hexlines ../QA/testfiles/simple1 -ShowPablo

bin/hexlines ../QA/testfiles/simple1 -trace=Byte