How I got sharp_combine working on a Macbook (system 10.5.5)

G. Novak, April 2009

I think the best way to proceed if you want to install sharp_combine on your mac is to follow the directions I've written below but also to rely heavily on "google" and on info from folks who have done it like John Vaillancourt and me. Also, it helps if you have experience with "c". So if you don't then you should write yourself a toy program (like "hello world") before starting. Finally, its good to know some unix or linux, though you don't need to know much.

To compile sharp_combine, I decided **not** to use Xcode, which is an "Integrated Development Environment". I decided to use the command-line compiler gcc instead.

However, it seemed like the easiest way to install gcc on my computer was to go ahead and install X-code, which automatically installs gcc.

Here is the e-mail from a local computer guru who's directions I followed to install Xcode:

... the latest version of the software is available for free from Apple if you'd like to download and install it yourself – you need to sign up for a free Apple Developer Connection account to download it <http://developer.apple.com/technology/Xcode.html>.

I think this went pretty well, as I recall.

The way to compile stuff in gcc these days is not via command-line invocation of gcc but rather via a makefile. The makefile does all the compiling, linking, and loading. Here is the makefile that I wrote and then used successfully, followed by John's makefile and then by the zamin makefile:

Giles' makefile:

```
CC     = gcc

HDRLIB  = /Users/novak/Pipeline/Libraries/include

LIBPATH = /Users/novak/Pipeline/Libraries/lib

LIBS    = -L$(LIBPATH) -lm -lcfitsio
```

```
CFLAGS  = -g -O2 -I$(HDRLIB)

OBJS= fitexy.o mnbrak.o zbrent.o nrutil.o chixy.o brent.o avevar.o fit.o gammq.o
gser.o gcf.o gammln.o erfcc.o

SRCS= fitexy.c mnbrak.c zbrent.c nrutil.c chixy.c brent.c avevar.c fit.c gammq.c gser.c
gcf.c gammln.c erfcc.c

IDLDIR=/Applications/itt/idl

BINDIR=$(IDLDIR)/bin/bin.darwin.i386

LDFLAGS= -no-cpp-precomp -dynamic -fno-common -L$(BINDIR) \

    -lidl -lXm -lMesaGL6_2 -lMesaGLU6_2 -lOSMesa6_2 -lfreetype2_1_3 \

    -multiply_defined suppress

sharp_combine:  sharp_combine.o dselect.o fit2.o chauvenet.o

        $(CC) $(CFLAGS) -o sharp_combine sharp_combine.o $(OBJS) dselect.o fit2.o
chauvenet.o $(LIBS) $(LDFLAGS)

clean:

        /bin/rm -f chauvenet.o fit2.o dselect.o sharp_combine.o
```

John's makefile:

```
CC     = gcc

HDRLIB  = /Users/johnv/Library/C

LIBPATH = /Users/johnv/Library/C

LIBS    = -L$(LIBPATH) -lm -lcfitsio -lnr

#CFLAGS  = -g -I$(HDRLIB)

CFLAGS  = -g -O2 -I$(HDRLIB)

IDLDIR=/Applications/rsi/idl

BINDIR=$(IDLDIR)/bin/bin.darwin.ppc

CALLTEST=$(IDLDIR)/external/callable/calltest
```

```
LDFLAGS= -no-cpp-precomp -dynamic -fno-common -L$(BINDIR) \
      -lidl -lXm -lMesaGL4_0 -lMesaGLU4_0 -lOSMesa4_0 -lfreetype2_1_3 \
      -llanginfo -multiply_defined suppress

sharpinteg:    sharpinteg.o
          $(CC) $(CFLAGS) sharpinteg.o $(LIBS) -o sharpinteg

sharpinteg_2:  sharpinteg_2.o
          $(CC) $(CFLAGS) sharpinteg_2.o $(LIBS) -o sharpinteg_2

quicklook:     quicklook.o
          $(CC) $(CFLAGS) quicklook.o $(LIBS) -o quicklook

all:          sharpsolve sharpstokes f_factor

f_factor:      f_factor.o select.o
          $(CC) $(CFLAGS) -o ffactor f_factor.o select.o $(LIBS)

fitgausspol:   fitgausspol.o
          $(CC) $(CFLAGS) -o fitgausspol fitgausspol.o $(LIBS)

fitgauss23:    fitgauss23.o
          $(CC) $(CFLAGS) -o fitgauss23 fitgauss23.o $(LIBS)

fitgauss2d:    fitgauss2d.o
          $(CC) $(CFLAGS) -o fitgauss2d fitgauss2d.o $(LIBS)
```

```
fitgauss:	fitgauss.o

	$(CC) $(CFLAGS) -o fitgauss fitgauss.o $(LIBS)


sharpsolve:	sharpsolve.o select.o

	$(CC) $(CFLAGS) -o sharpsolve sharpsolve.o select.o $(LIBS)


sharp_combine_v4:  sharp_combine_v4.o dselect.o fit2.o chauvenet.o

	$(CC) $(CFLAGS) -o sharp_combine_v4 sharp_combine_v4.o dselect.o fit2.o
chauvenet.o $(LIBS) $(LDFLAGS)


sharp_combine:  sharp_combine.o dselect.o fit2.o chauvenet.o

	$(CC) $(CFLAGS) -o sharp_combine sharp_combine.o dselect.o fit2.o
chauvenet.o $(LIBS) $(LDFLAGS)


sharp_combine_old:  sharp_combine_v362.o dselect.o fit2.o

	  $(CC) $(CFLAGS) -o sharp_combine_old sharp_combine_v362.o dselect.o
fit2.o $(LIBS)


sharpstokes:	sharpstokes.o

	$(CC) $(CFLAGS) -o sharpstokes sharpstokes.o $(LIBS)


sharp_qu:	sharp_qu.o

	$(CC) $(CFLAGS) -o sharp_qu sharp_qu.o $(LIBS)


sharpsum:	sharpsum.o

	$(CC) $(CFLAGS) -o sharpsum sharpsum.o $(LIBS)

sharpvis:	sharpvis.o

	$(CC) $(CFLAGS) -o sharpvis sharpvis.o $(LIBS) -L/usr/X11R6/lib -lGL -
bind_at_load -framework GLUT -framework Cocoa
```

```
testchauv:	testchauv.o chauvenet.o
	$(CC) $(CFLAGS) -o testchauv testchauv.o chauvenet.o $(LIBS)
chauvenet:	chauvenet.o
	$(CC) $(CFLAGS) -o chauvenet chauvenet.o $(LIBS)

clean:
	/bin/rm -f *.o
```

The zamin makefile:

```
#
# Makefile for SHARP reduction programs
#
CC=gcc
#
# Henry added include directory for fitsio routines
#
CFLAGS=-g -O2 -I/usr/local/cfitsio/cfitsio

#CFLAGS=-g -O2
LIBS=-lm -lcfitsio
OBJS= FITEXY.o MNBRAK.o ZBRENT.o NRUTIL.o CHIXY.o BRENT.o AVEVAR.o FIT.o
GAMMQ.o GSER.o GCF.o GAMMLN.o erfcc.o
SRCS= FITEXY.c MNBRAK.c ZBRENT.c NRUTIL.c CHIXY.c BRENT.c AVEVAR.c FIT.c
GAMMQ.c GSER.c GCF.c GAMMLN.c
```

```
COROBJS=BRENT.o f1dim.o linmin.o MNBRAK.o NRUTIL.o powell.o
CORSOR=BRENT.c f1dim.c linmin.c MNBRAK.c NRUTIL.c powell.c
SHOBJS= select.o gaussj.o realft.o four1.o


IDLDIR=/usr/local/itt/idl
#IDLDIR=/usr/local/rsi/idl
#IDLDIR=/opt/rsi/idl
IDLBINDIR=$(IDLDIR)/bin/bin.linux.x86_64
LDFLAGS= -L$(IDLBINDIR) -lidl -Wl,-rpath,. -Wl,-rpath $(IDLBINDIR)\
      /usr/lib64/libXm.a \
      -lXp -L/usr/lib64 -lXmu -lXext -lXt -lSM -lICE -lX11 -ldl \
      -lrt -lm -lpthread \
      -ltermcap /usr/lib/libXpm.so.4 -lXinerama
#      -lgcc_s \


sharcslice:    sharcslice.o
          $(CC) $(CFLAGS) sharcslice.o $(LIBS) -o sharcslice
sharctau:      sharctau.o
          $(CC) $(CFLAGS) sharctau.o $(LIBS) -o sharctau

sharptau:      sharptau.o
          $(CC) $(CFLAGS) sharptau.o $(LIBS) -o sharptau

chi2:   chi2.o
      $(CC) $(CFLAGS) chi2.o $(LIBS) -o chi2
```

```
sharpinteg:     sharpinteg.o $(OBJS)
        $(CC) $(CFLAGS) sharpinteg.o $(OBJS) $(LIBS) -o sharpinteg

sharpinteg_2:   sharpinteg_2.o $(OBJS)
        $(CC) $(CFLAGS) sharpinteg_2.o $(OBJS) $(LIBS) -o sharpinteg_2

sharp_combine_v5:  sharp_combine_v5.o fit2.o dselect.o chauvenet.o
        $(CC) $(CFLAGS) -o sharp_combine_v5 sharp_combine_v5.o $(OBJS) fit2.o
dselect.o chauvenet.o $(LIBS) $(LDFLAGS)

sharp_combine:  sharp_combine.o fit2.o dselect.o chauvenet.o
        $(CC) $(CFLAGS) -o sharp_combine sharp_combine.o $(OBJS) fit2.o dselect.o
chauvenet.o $(LIBS) $(LDFLAGS)

correlator:     correlator.o $(COROBJS)
        $(CC) $(CFLAGS) -o correlator correlator.o $(COROBJS) $(LIBS)

sharpsolve:     sharpsolve.o $(SHOBJS)
        $(CC) $(CFLAGS) -o sharpsolve sharpsolve.o $(SHOBJS) $(OBJS) $(LIBS)

testchauv:      testchauv.o ran1.o chauvenet.o erfcc.o $(OBJS)
        $(CC) $(CFLAGS) -o testchauv testchauv.o ran1.o chauvenet.o erfcc.o
$(OBJS) $(LIBS)
```

Using the makefile to compile sharp_combine:

Make a directory where you will keep your source code and put the makefile in it. When you want to compile, just type "make clean" followed by "make sharp_combine".

You need the latest version of sharp_combine.c, as well as John's dselect.c fit2.c and chauvenet.c.

Also, there are three other sources of code that you need to get going: fitsIO stuff, numerical recipes routines, and IDL routines (for automatic display of background removal diagnostics). Here is how I got these working:

Fitsio installation:

I installed the fitsio routines as a dynamic or shared library. You get this stuff from the following web page:

http://heasarc.gsfc.nasa.gov/fitsio/

Follow the instructions given under the heading "How to compile CFITSIO as a Universal Binary on mac OS-X". Note that I ignored the first set of instructions since they were for X-code and instead I used the second set, that are under the sub-heading "Another way to build the universal binary":

Another way to build the universal binary:

1. unpack the cfitsio source code tar file

2. cd cfitsio

3. setenv CFLAGS "-arch ppc -arch i386 -g -O2"

4. Then proceed with the standard cfitsio build, i.e.:
    * ./configure

* make

* make install

After consulting with “google” and also with the more general but more thorough “unix” instructions, that can be accessed by clicking on: “CFITSIO Quick Start Guide” also on this NASA site, I refined the above instructions as follows:

./configure

should be replaced by

./configure --prefix=/Users/novak/Pipeline/Libraries

make

should be replaced by

make shared

Since I use “bash”, the setenv command should be replaced by the “bash equivalent” (google it)

The above procedure puts fitsio library stuff in two subdirectories of the above-referenced “Libraries” directory. Next I downloaded and compiled a test program just to be sure I could read the headers of random fits files using this fitsio library that I just installed. This was the makefile I used for that test program I got from the site:

```
CC     =      gcc

HDRLIB  =      /Users/novak/Pipeline/Libraries/include

LIBPATH =      /Users/novak/Pipeline/Libraries/lib

LIBS    =      -L$(LIBPATH) -lm -lcfitsio
```

```
CFLAGS  =       -g -O2 -I$(HDRLIB)


fits_test:      fits_test.o
        $(CC) $(CFLAGS) -o fits_test fits_test.o $(LIBS)
```

Note that since I am using a dynamic library I had to add the following line to my .bash_profile file:

export DYLD_LIBRARY_PATH=/Applications/itt/idl/bin/bin.darwin.i386:/Users/novak/Pipeline/Libraries/lib

(The first item in the above path is actually for the IDL stuff that comes next, the second item is the one that is needed if you are to run a dynamic fitsio library.)

IDL installation:

I already had IDL installed on my macbook. (I did this a year ago and can't remember how hard or easy it was, though I probably have some old notes on this.)

I started by compiling a test program that was in a directory called "call test". The readme file in that directory seemed to have pretty clear instructions on how to get that working. I managed to compile it and get it working using the following Makefile, which I think I copied verbatim from what was described and provided and the "call test" directory:

```
IDLDIR=/Applications/itt/idl

BINDIR=$(IDLDIR)/bin/bin.darwin.i386

CALLTEST=$(IDLDIR)/external/callable/calltest

LDFLAGS= -no-cpp-precomp -dynamic -fno-common -L$(BINDIR) \

    -lidl -lXm -lMesaGL6_2 -lMesaGLU6_2 -lOSMesa6_2 -lfreetype2_1_3 \

    -multiply_defined suppress
```

```
idl.new :
	cc -o idl.new $(BINDIR)/main.o $(LDFLAGS)
	rm -f idldе.new; ln idl.new idldе.new



# Build the C language calltest program.
calltest.o : $(CALLTEST).c
	cc -I$(IDLDIR)/external/include -c $(CALLTEST).c


calltest : calltest.o
	cc calltest.o -o calltest $(LDFLAGS)
```

Note: according to my notes there are three environment variables you need to define, but for some reason I only made records about two of them: From my .bash_profile file:

export IDL_DIR=/Applications/itt/idl

export DYLD_LIBRARY_PATH=/Applications/itt/idl/bin/bin.darwin.i386:/Users/novak/Pipeline/Libraries/lib

The third may not have been important, but in any case I think its all documented in the instructions and examples I found in that "call test" directory.

Note that the instructions are geared toward Xcode but are mixed in with "unix-specific instructions". As I recall, it was pretty easy to figure out what to do. I ignored the Xcode instructions that were specifically geared toward stuff you were supposed to do in the Xcode environment since I don't work in that environment. I think I basically ended up following the "unix specific instructions" in the README.

With this info, plus following John's makefile, I was able to figure out how to write the IDL-specific stuff in my makefile that is given above.

Numerical recipes installation:

I actually paid the $75 to legally download the numerical recipes routines from their site. John built himself a library of numerical recipes, but all I did was to copy the 13 needed C routines and the one header file (nrutil.h) into my source directory. The 13 routines you need are listed in my makefile. In doing this I was following the way its done on zamin.

This did not quite work. I still had to "hand compile" the 13 routines using

gcc –g –O2 …etc.

That makes the ".o" files and then you never have to do this again once they are made.

Testing the code:

Posted to the SHARP analysis logbook there is a link called "test-for-sharp_combine" that lets you download a tar file which includes 12 sharp_integ output files, one list-file, and one text file showing all the sharp_combine flags that should be invoked.

This can all be used together to test sharp_combine to be sure that its working. If its working then the Q-map should have the following value for pixel (27,27): 2.75 E-5

The first time I ran the code it crashed. John and I had to make some changes to some of the fitsIO stuff to get it to run. Specifically we removed "insert" commands, replacing them with "write" commands. It seems like this is some sort of "platform dependent" thing that only causes trouble on an Intel Macbook like mine. In any case, this will be fixed in the next version of sharp_combine.